Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Čepar

Razširitve CMS z lastnimi moduli

diplomska naloga
na univerzitetnem študiju

Mentor: doc. dr. Andrej Brodnik

Ljubljana 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge: Razširitve CMS z lastnimi moduli

V diplomski nalogi na kratko preglejte področje CMS (Content Managment System) s poudarkom na CMS Magnolia. V pregledu primerjajte CMS Magnolia z drugimi primerljivimi sistemi ter opredelite razloge, da bi se osredotočili na uporabo le-te.

Posebej se osredotočite na možnost razširitve CMS s svojimi lastnimi moduli. Zatorej opišite notranjo arhitekturo Magnolie in format podatkovja ter način dodanja novih modulov. Kot primer razširitve razvijte tri samostojne module za Magnolio. Moduli naj omogočijo internacionalizacijo statičnih strani, podporo za navidezne spletne strežnike neposredno iz CMS ter učinkovitejšo podporo za vnos podatkov preko obrazcev.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matjaž Čepar, z vpisno številko **63040021**, sem avtor diplomskega dela z naslovom:
Razširitve CMS z lastnimi moduli

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Andreja Brodnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 10. maja 2014

Podpis avtorja:

Zahvaljujem se družini za podporo in potrpljenje med študijem.

Obenem se zahvaljujem vsem profesorjem in sošolcem za pomoč pri pridobivanju znanja. Rad bi se zahvalil tudi podjetju Parsek, d. o. o., predvsem Andreju Rakovcu za mentorstvo in učenje ogrodja Magnolia.

Ob koncu se zahvaljujem še svojemu mentorju Andreju Brodniku za pomoč pri pisanju diplomskega dela.

Kazalo

Povzetek

Cilj diplomske naloge je izdelava treh razširitev za prosto dostopno različico sistema Magnolia CMS.

V prvem delu diplomske naloge smo predstavili sistem Magnolia in se osredotočili predvsem na njegov zaledni del. Natančneje smo opisali podatkovni model in način izdelave razširitev.

V drugem delu opisujemo razširitve. Za diplomsko nalogo smo izdelali tri razširitve. Razširitev za lokalizacijo statičnih vsebin nam omogoča spreminjanje delov spletne strani, do katerih nimamo neposrednega dostopa. Razširitev za gostovanje več domen nam omogoči, da znotraj ene aplikacije živijo spletne strani, ki se nahajajo na več različnih domenah, vsaka s svojo vsebino. Razširitev za shranjevanje podatkov iz obrazcev nam omogoči, da zaobidemo težavo, ki jo povzroča ločenost sistema na dve ali več podatkovnih baz.

Rezultat diplomske naloge so tri delujoče razširitve, ki jih lahko nemudoma in neodvisno uporabimo v svojem lastnem Magnolia projektu.

Ključne besede: Magnolia, CMS, Java, podatkovne baze.

Abstract

The goal of thesis was to develop extensions for Magnolia CMS Community edition.

The first part focuses on Magnolia itself. It consists of a quick introduction to Magnolia CMS and primarily focuses on its back-end. The database system and how to develop extensions were described in detail.

The second part focuses on actual extensions. We have developed three extensions for this thesis. Static localization extension enables us to change content, that would otherwise be inaccessible. The multi-site extensions allow the application to contain independent content from multiple domains. The form module extension solves the problems that are caused by having at least two separate databases.

The results of this thesis are three independent modules, that can be independently used when developing a Magnolia application.

Keywords: Magnolia, CMS, Java, databases

Poglavje 1

Uvod

Svetovni splet je kaotičen. S tem ne mislimo samo na navadnega uporabnika, ampak na vse vpletene. Navadni uporabniki se morajo soočati z različnimi tehnologijah, o katerih ne vedo nič, v idealnem svetu pa jim tudi ne bi bilo treba nič vedeti. Ljudje, ki ustvarjajo splet, imajo iste težave, vendar pa morajo poznati še tehnologije. Čeprav je tehnologij zelo veliko, so vse usmerjene k zadovoljstvu končnega uporabnika. Izkušeni ustvarjalci so kmalu ugotovili, da večino časa uporabljajo ene in iste tehnologije na en isti način. Tako so se počasi razvile posplošene rešitve, ki si jih lahko z minimalnimi spremembami pretvoril v nekaj, kar ustreza lastnim potrebam. Imenujejo se ogrodja ali angleško »frameworks«.

To pa ni bilo dovolj. Različni ljudje različno pristopamo k istim težavam in tako stalno nastajajo in izginjajo različna ogrodja. Učinkovitost na tem področju računalništva se zato ne meri samo v suhoparnih podatkih, kot so odzivni čas, O-notacija, poraba pomnilnika itd. Ti podatki so še vedno pomembni, pri tem pa je pomembno še, koliko časa je rešitev na trgu in v aktivni uporabi. Na področju, kjer se izdelki stalno rojevajo in umirajo, je dolga življenjska doba načeloma znak kakovosti. V tem delu se bomo večinoma ukvarjali s sistemom Magnolia CMS, ki aktivno živi od leta 2003. Najprej bomo predstavili nekaj neposrednih tekmecev in tehnologij, na katerih je zgrajena Magnolia.

1.1 Motivacija

Magnolia je CMS-sistem, primarno namenjen za razvoj spletnih aplikacij. Obstaja v prosto dostopni odprtokodni različici in v dveh plačljivih različicah. Jedro je v obeh primerih enako. Plačljive različice vsebujejo dodatne priboljške, kot so napredno upravljanje predpomnilnika, podpora za večdomensko spletno aplikacijo, zmožnost namestitve na IBM-ov WebSphere strežnik itd.

Magnolia je zelo modularno zasnovana, tako da lahko sami spišemo vsakršno rešitev, tudi takšno, ki ne obstaja v nobeni različici. Modularnost nam prav tako omogoča zamenjavo vsake komponente sistema s poljubno lastno rešitvijo. Ker prosta različica ne omogoča vseh potrebnih funkcionalnosti, ki jih potrebujemo za vsakodnevno uporabo, smo bili primorani razviti svoje rešitve.

1.2 Cilji

Cilj diplomske naloge je spisati dodatne module za reševanje pogostih težav, s katerimi se srečujemo spletni razvijalci. Pri opisu vsake rešitve bomo podali tudi kratek opis glavnih tehnologij in protokolov, ki so potrebni za razumevanje funkcionalnosti modulov. Ker o modulih ne moremo povedati nič, brez da bi razumeli osnovo, bomo tudi zelo na kratko predstavili Magnolio in miselni proces pri njenem razvoju in prilagajanju. Ravno tako bomo na hitro omenili nekaj splošno poznanih spletnih CMS-ov in poskušali oceniti, kam se uvršča Magnolia. V diplomski nalogi bomo rešili tri konkretne težave, s katerimi se bo zagotovo srečal skoraj vsak uporabnik sistema Magnolia:

- lokalizacija samodejno ustvarjenih vsebin: tem vsebinam bomo pravili statične vsebine, saj uporabnik nima neposrednega nadzora nad njimi;
- upravljanje neodvisnih spletnih strani znotraj ene aplikacije: ena namestitev aplikacije Magnolia lahko vsebuje vsebine, ki so vezane na točno določene domene. Tukaj smo razvili modul, ki je sposoben iz podatka o domeni določiti, katera vsebina ji pripada.;

- shranjevanje podatkov iz spletnih obrazcev: to je na prvi pogled zelo trivialno opravilo, vendar se zaradi posebnih lastnosti Magnolie izkaže kot velika težava.

Poglavje 2

Tehnologije

V tem poglavju bomo na kratko predstavili tehnologije, ki smo jih uporabljali pri razvoju. V diplomskem delu jih nismo nikjer podrobno razložili. Za lažje razumevanje naloge je priporočeno, da jih bralec vsaj v osnovi pozna.

2.1 Razvojno okolje

2.1.1 Programski jezik Java

Java je objektno usmerjen programski jezik. Je eden izmed najbolj razširjenih jezikov. Idejni avtor je James Gosling, ki ga je razvil v podjetju Sun Microsistems. Leta 2010 je Sun Microsistems prevzel Oracle, ki še sedaj skrbi za Javo.

Java je bila že v začetku zamišljena kot programski jezik, ki bi bil neodvisen od platforme. To pomeni, da program, ki je bil razvit na eni platformi, brez sprememb v kodi deluje na vseh platformah. Ker brez vmesnika med programom in platformo to ni mogoče, je bila razvita JVM (angl. Java Virtual Machine). JVM je odvisna od platforme (za vsako platformo obstaja svoja različica JVM), ki poganja Java programe. Vsi javanski programi se prevedejo v kodo »Java byte code«, ki je enaka za vse različice JVM.

Z različico Java 2 je Java začela izhajati v različnih konfiguracijah. Vsaka konfiguracija je prilagojena določenemu namenu in poleg glavnih javanskih

knjižnic vsebuje še dodatne razširitve. Uporabnik lahko z vključevanjem in odstranjevanjem knjižnic poljubno spreminja konfiguracije. Java SE je standardna različica, ki jo ima nameščeno večina namiznih in prenosnih računalnikov. Java ME je bila namenjena razvoju na mobilnih napravah. Java EE cilja predvsem na zahtevnejše spletne aplikacije. Magnolia je napisana v Java EE, kar pomeni, da je napisana v programskem jeziku Java in uporablja veliko API-jev, ki pridejo z Java EE.

2.1.2 Jboss AS

Jboss je aplikacijski strežnik, ki temelji na Java EE. Aplikacijski strežnik je strežnik, ki aplikacijo poganja, hkrati pa tudi omogoča dostop do raznih knjižnic, ki so že prednameščene na strežniku. Za aplikacijske strežnike bi lahko rekli, da brišejo mejo med tradicionalnimi strežniki in ogrodji.

Jboss ima vgrajen spletni strežnik Tomcat. Na njem se izvajajo »servleti« in JSP-strani. To omogoča dinamično generiranje vsebin glede na potrebe odjemalca. Obstaja v dveh različicah, to sta prosto dostopna različica Jboss Community Edition in komercialna Jboss Enterprise edition. Jboss smo izbrali, ker je zmogljiv, brezplačen in preverjeno deluje z Magnolia CMS.

2.1.3 Apache Maven

Maven je orodje, ki v osnovi avtomatizira proces prevajanja izvorne kode. Je pa tudi veliko več. Za Maven obstaja veliko različnih vtičnikov, ki zelo poenostavljajo delo s projekti. Poleg prevajalnika smo uporabili še naslednje vtičnike:

- JRebel: ta vtičnik avtomatsko generira datoteke, ki strežniku povejo, da je aplikacija še v razvoju in kje so izvorne datoteke. Strežnik nato izvaja izvorne datoteke, kar močno pohitri razvoj, ker ni potrebno ponovno nameščati aplikacije na strežnik
- Jboss deployment: ta vtičnik samodejno zazna delujoč strežnik in nanj namesti aplikacijo

- War: vtičnik prevedeno aplikacijo avtomatično zapakira v standardno WAR datoteko. WAR datoteka je JAR datoteka, s točno določeno strukturi, ki jo spletni strežniki prepoznajo in avtomatsko zaženejo.

Celoten projekt Maven se nastavlja preko POM-datotek. V POM-datoteke zapišemo module, vtičnike in povezave med njimi. Moduli so logične enote. Lahko so zunanje knjižnice ali samo njihovi posamezni deli. Lastna aplikacija je tudi en ali več modulov. Med seboj so poljubno povezani v drevesne strukture. Maven pri prevajanju upošteva odvisnosti med posameznimi moduli. Najprej prevede module, ki nimajo nobenih odvisnosti, potem pa postopoma prevaja vse module, ki imajo prevedene tudi vse module, od katerih so odvisni, dokler ni prevedena celotna aplikacija. Sposoben je tudi avtomatskega prenosa že obstoječih modulov z interneta, če mu v POM-datoteki ustrezno nastavimo lokacije repozitorijev.

2.1.4 PostgreSQL 9.0

PostgreSQL je odprtokodni sistem za upravljanje s podatkovnimi bazami. Omogoča standardne operacije na relacijskih bazah (sprožilci, sekvence, pogledi, indeksi, tuji ključi itd.). Vgrajen ima tudi proceduralni jezik PL/pgSQL, ki služi izvajanju raznih procesov na podatkovni bazi.

2.2 Ogrodja za spletne strani

Na svetu obstaja veliko rešitev za izdelavo spletnih vsebin. V tem poglavju bomo omenili in na hitro predstavili nekaj največjih in najbolj popularnih.

Na tem mestu bi pojasnili, kaj sploh je CMS. CMS (angl. content management sistem) je vmesnik za ustvarjanje in objavo vsebin. Vse Magnolia CMS in vse v nadaljevanju naštete rešitve niso samo ogrodja za izdelavo spletnih strani, ampak vsebujejo tudi spletno aplikacijo za vnos in upravljanje z vsebinami. Take aplikacije imenujemo spletni CMS.

2.2.1 WordPress

WordPress (WP) je odprtokodni projekt, ki je napisan v programskem jeziku PHP in uporablja MySQL podatkovno bazo (https://wordpress.org/). Primarno je namenjen za bloge, vendar lahko deluje tudi kot CMS. Ustvarila sta ga Matt Mullenweg in Mike Little leta 2003.

V WP-ju se ni treba ukvarjati s standardnim razvojem HTML-strani, ampak razvijamo teme in vtičnike. V splošnem še to ne, ker obstaja izvrstna skupnost, ki je izdelala že veliko razširitev in jih ponudila v brezplačno uporabo. Izdelava spletne strani v WP-ju je zato zelo hitra. Izberemo temo, želene vtičnike in vnesemo vsebine. To pa ima tudi negativne posledice.

Če delamo kakšno večjo stran, bomo zagotovo potrebovali novo temo in verjetno tudi kakšen nov vtičnik, izdelava teh pa je časovno potratna. Ker je WP zasnovan kot blog, ima vsaka predloga zgolj en del za vsebino. Če želimo vsebino razdeliti preko več delov HTML-strani, moramo uporabiti programerske trike, kar pa zelo poveča čas in zahtevnost razvoja ter vzdrževanja aplikacije.

WP-ja se drži sloves izredno ranljive aplikacije. V zadnjem času so se izredno popravili, vendar če pogledamo javno odprt seznam še neodpravljenih težav, vidimo, da jih je veliko resnejše narave.

2.2.2 Drupal

Drupal je odprtokodni projekt v programskem jeziku PHP. Uporablja lahko praktično katero koli relacijsko bazo (https://drupal.org/). Prva različica je izšla leta 2001. Trenutno nad projektom bdi Drupal Association, ki strateško vodi projekt, razvija pa ga skupnost.

Na prvi pogled je podoben WordPressu, vendar je njegova filozofija popolnoma drugačna. WP uporabnike zaradi težavnosti razvoja sili v uporabo predpripravljenih rešitev, Drupal pa omogoča hitro izdelavo lastnih tem in rešitev, s čimer pa izgubimo prenosljivost komponent med različnimi projekti. V primerjavi z WP-jem ima strmejšo učno krivuljo.

2.2.3 Joomla

Joomla je odprtokodni projekt, spisan v programskem jeziku PHP (http://www.joomla.org/). Uporablja lahko podatkovne baze MySQL, MS SQL ali PostgreSQL. Izšla je leta 2005.

Joomla poskuša združiti najboljše iz WP-ja in Drupala. Vsebuje veliko že predpripravljenih tem, podobno kot WP, razvoj novih tem pa je lažji kot v WP-ju, podobno kot v Drupalu. Posledica tega je, da ne vsebuje toliko tem kot WP in razvoj ni tako enostaven kot v Drupalu, ampak je nek vmesni člen med obema.

2.2.4 LifeRay

LifeRay je ogrodje za izdelavo spletnih portalov (https://www.liferay.com/). Portal je spletna stran/aplikacija, ki se po videzu in funkcionalnosti prilagaja trenutnemu obiskovalcu. Primer portala bi bila časopisna stran, ki neprijavljenemu obiskovalcu prikazuje vsebine. Če se novinar prijavi vanjo, namesto vsebin vidi urejevalnik besedil, v katerega lahko vpiše članke, urednik pa ima na voljo seznam člankov, ki bi jih lahko kadar koli objavil ali umaknil s strani.

LifeRay je spisan v Java EE in lahko uporablja praktično katero koli relacijsko bazo. Avtor je podjetje LifeRay, Inc., ki je bilo ustanovljeno leta 2000.

LifeRay obstaja v dveh različicah: Liferay Portal Community Edition in Liferay Portal Enterprise Edition. CE je v celoti odprtokodna, medtem ko EE vsebuje tudi zaprtokodne razširitve. EE ponuja tudi boljšo dokumentacijo in podporo.

LifeRay je že v začetku namenjen za kompleksnejše spletne strani, kjer se tudi dobro izkaže. Ker pa ima že privzeto veliko število modulov in funkcionalnosti, je razvoj na njem počasnejši in porablja veliko resursov. Ima tudi zelo strmo učno krivuljo. To ga naredi neprimernega za manjše projekte.

2.2.5 Povzetek

Obstaja še veliko različnih ogrodij. V tem podpoglavju smo na hitro predstavili nekaj najbolj razširjenih in prepoznavnih ter konceptualne razlike med njimi.

Ogrodja so zgolj orodja za reševanje specifičnih problemov. Nobeno od njih ne rešuje vseh težav. Uporabnik mora sam dobro poznati svojo težavo in izbrati najbolj ustrezno orodje. V tej diplomski nalogi smo se za Magnolijo odločili iz več razlogov. Magnolijo smo uporabili pri več projektih in zelo dobro poznamo njene prednosti in slabosti pri praktični uporabi. Magnolija ravno tako ne uporablja relacijske podatkovne baze, ampak drevesno. V primerjavi z relacijskimi podatkovnimi bazami so drevesne malo poznane in uporabljane, zato jih bomo tekom diplomske naloge podrobneje opisali in izpostavljali, kako jih Magnolija in naši moduli uporabljajo.

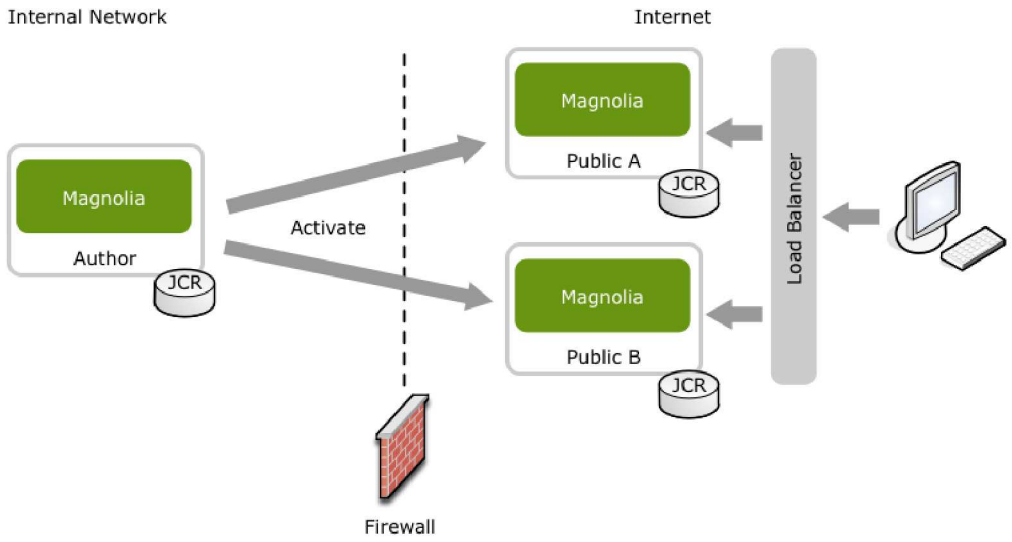
Poglavje 3

Magnolia CMS

Magnolia CMS je v Javi EE spisan sistem za poenostavljanje objav spletnih vsebin na spletne strani[1] – spletni CMS. Obstaja v dveh različicah: odprtokodni Community edition in z zaprtokodnimi moduli razširjeni Enterprise Edition. Razvija ga Magnolia International Ltd., prva različica je izšla 15. novembra leta 2003.

Od večine ostalih sistemov se razlikuje po tem, da je že v osnovi zasnovana za več stražnikov, ampak ne samo v klasičnem porazdeljenem smislu (slika 3.1). Vedno naj bi imeli vsaj eno avtorsko instanco in eno javno instanco. V avtorsko instanco se prijavljajo administratorji, to so ljudje, ki lahko urejajo vsebino, skrbijo za konfiguracijo, varnostne nastavitve itd. Vse te spremembe pa se potem objavljajo v javni instanci, ki je namenjena končni uporabnikom (slika 3.1).

Druga nestandardna lastnost je uporaba drevesne podatkovne baze namesto klasične relacijske. Konkretno uporablja referenčno implementacijo standarda JSR 170 – Apache Jackrabbit. Magnolia omogoča tudi integracijo s poljubnimi bazami. Ker brez razumevanja drevesnih podatkovnih baz Magnolie ne moremo ne uporabljati ne razumeti, je v naslednjem poglavju vključena kratka predstavitev tovrstnih baz. Predstavitev relacijskih baz smo vključili zaradi lažje primerjave in predstave, kako se ta dva koncepta razlikujeta, in ker ena izmed predstavljenih rešitev uporablja integracijo z



Slika 3.1: Primer enostavne topologije s tremi strežniki[2]

relacijsko bazo.

3.1 Podatkovne baze

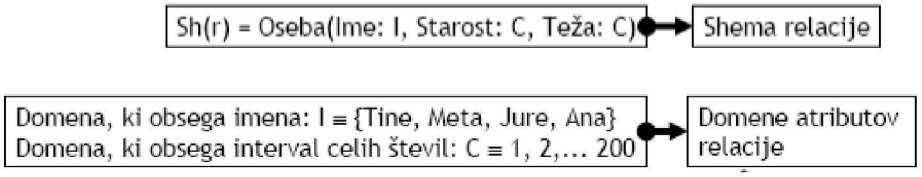
Kot smo že omenili, se Magnolia CMS od ostalih rešitev razlikuje predvsem v tem, da ne uporablja standardnega relacijskega modela za shranjevanje podatkov, ampak drevesnega. V tem poglavju zato predstavimo relacijski in objektni model ter na koncu na kratko še razliko med njima.

3.1.1 Relacijski model

3.1.1.1 Model

Relacijski model izhaja iz teorije množic[3]. Pri njih uporabljamo določene izraze: relacija, atribut, domena, n-terica, relacijska podatkovna baza.

Relacijo si lahko predstavljamo kot dvodimenzionalno tabelo s stolpci in vrsticami. Atribut je stolpec v tabeli – relaciji. Domena je množica vseh dovoljenih vrednosti atributa, ki pripada tej domeni. N-terica je ena vrstica v relaciji. Relacijska podatkovna baza je množica relacij z unikatnimi imeni.



Slika 3.2: Enostavna shema in domeni za relacijo »Oseba«

Vsaki relaciji pripada ustrezna relacijska shema. S shemo (Sh) konceptualno opišemo posamezno relacijo. Vsaki relaciji pripada natanko ena shema, ena shema pa lahko pripada več relacijam. Enostavna shema je podana v sliki 3.2.

Relacije imajo še nekaj dodatnih lastnosti in omejitev:

- celice lahko vsebujejo samo atomarne vrednosti,
- vsak atribut v posamezni relaciji ima svoje unikatno ime,
- v posamezni relaciji se n-terice ne podvajajo,
- vrstni red atributov in n-teric ni pomemben.

Ker se n-terice ne smejo pojavljati, lahko uvedemo pojem ključev. Ključi so množice atributov relacije s točno določenim logičnim pomenom. Obstajajo naslednje vrste ključev:

- kandidat za ključ je vsaka množica atributov n-terice, ki enolično določa n-terico znotraj posamezne relacije,
- primarni ključ je izbrani kandidat za ključ,
- nadključ je vsaka množica atributov, ki vsebuje primarni ključ,
- tuji ključ je primarni ključ neke druge ali iste relacije, ki služi za logično povezovanje relacij.

Vsak načrtovalec podatkovne baze se mora zavedati še nekaj dodatnih omejitev. Vsaka n-terica mora vsebovati primarni ključ. Če imajo relacije tuje

D	E
D1	E1
D2	E2

Tabela 3.1: Relacija A

D	E
D2	E2
D3	E3

Tabela 3.2: Relacija B

ključe, potem morajo ti nujno imeti vrednost primarnega ključa povezane relacije ali pa morajo biti prazni (null). Načrtovalci lahko poljubno dodajajo še svoje lastne omejitve, vendar so zgoraj naštete neobhodne.

3.1.1.2 Dostop do podatkov

Do podatkov iz relacijskega modela dostopamo z relacijsko algebro, ki vsebuje pet osnovnih operacij: selekcija, projekcija, kartezijski produkt, unija in razlika. Možne so tudi izpeljane operacije, najbolj znana med njimi je stik. Za lažjo predstavo bomo tudi prikazali vse operacije s pomočjo tabel 3.1, 3.2 in 3.3.

Selekcija deluje na eni relaciji in vrne relacijo, ki vsebuje samo tiste n-terice, ki ustrezajo izbranemu pogoju. Primer je podan v enačbi 3.1.

E	F
E2	F2
E3	F3
E4	F4
E5	F5

Tabela 3.3: Relacija C

$$\sigma_{D<D2}(A) = \begin{bmatrix} D1 & E1 \end{bmatrix} \tag{3.1}$$

Projekcija deluje na eni relaciji in vrne relacijo, ki vsebuje samo izbrane stolpce (atribute). Operacija tudi samodejno odstrani možne podvojene n-terice. Primer je podan v enačbi 3.2.

$$\pi_D(A) = \begin{bmatrix} D2 \\ D3 \end{bmatrix} \tag{3.2}$$

Unija deluje na dveh relacijah, ki imata identične atribute. Rezultat unije je relacija, ki vsebuje vse n-terice iz obeh relacij. Tudi unija odstrani duplikate. Primer je podan v enačbi 3.3

$$A \cup B = \begin{bmatrix} D1 & E1 \\ D2 & E2 \\ D3 & E3 \\ D4 & E4 \end{bmatrix} \tag{3.3}$$

Razlika deluje med dvema relacijama, ki se ujemata v atributih. Razlika med relacijama A in B vrne relacijo, ki vsebuje vse n-terice, ki so v A in jih ni v B. Kartezijski produkt med dvema relacijama vrne vse možne kombinacije med n-tericami iz obeh relacij. Primer je podan v enačbi 3.4

$$A - B = \begin{bmatrix} D1 & E1 \end{bmatrix} \tag{3.4}$$

Kartezijski produkt med dvema relacijama vrne vse možne kombinacije med n-tericami iz obeh relacij. Primer je podan v enačbi 3.5

$$A\,x\,B = \begin{bmatrix} D1 & E1 & D2 & E2 \\ D1 & E1 & D3 & E3 \\ D2 & E2 & D2 & E2 \\ D2 & E2 & D3 & E3 \end{bmatrix} \tag{3.5}$$

Zelo pomembna operacija je **stik**. Stik nam omogoča enostaven pregled podatkov iz med seboj povezanih relacij. Je tudi računsko izredno zahtevna

operacija in je običajno najbolj časovna potratna operacija pri poizvedbah v bazo. Če združimo kartezični produkt s selekcijo v eno operacijo, dobimo stik. Obstaja več vrst stikov: θ-stik, Naravni stik, Odprti stik, Delni stik.

θ-stik (enačba3.6) med relacijama A in B vrne relacijo vseh n-teric kartezičnega produkta A in B, ki ustreza predikatu P. P je oblike A [predikat] B, kjer je [predikat] primerjalna operacija.

$$B \bowtie_F C;\ F = E_{(B)} < E_{(C)} = \begin{bmatrix} D2 & E2 & E3 & F3 \\ D2 & E2 & E4 & F4 \\ D2 & E2 & E5 & F5 \\ D3 & E3 & E4 & F4 \\ D3 & E3 & E4 & F4 \end{bmatrix} \tag{3.6}$$

Naravni stik (enačba 3.7) med A in B je oblika θ-stika, ki se avtomatsko združi po skupnih atributih med relacijama in operacija v predikatu P je enakost.

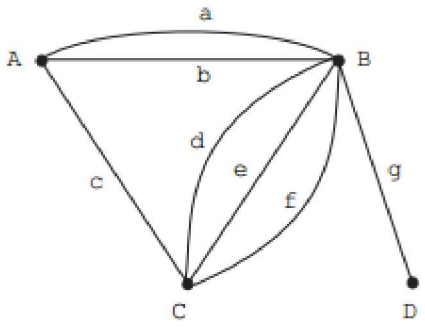
$$B \bowtie C = \begin{bmatrix} D2 & E2 & F2 \\ D3 & E3 & F3 \end{bmatrix} \tag{3.7}$$

Zunanji stik je podoben θ-stiku, le da ohrani vnose, ki nimajo vrednosti v stičnem atributu. Poznamo levo-odprti (enačba 3.8) in desno-odprti zunanji stik. Zunanji stik med relacijama A in B je levo-odprti, če ohrani vse n-terice iz relacije A. Če ohrani vse n-terice iz relacije B, je desno-odprti.

$$(C \triangleright_F B;\ F = (E_{(C)} = E_{(B)})) = \begin{bmatrix} E2 & F2 & D2 & E2 \\ E3 & F3 & D3 & E3 \\ E4 & F4 & & \\ E5 & F5 & & \end{bmatrix} \tag{3.8}$$

3.1.2 Mrežni model

Mrežni model temelji na teoriji grafov[4]. Magnolia CMS uporablja drevesni model. Mrežni model omenjamo, ker je drevesni model izpeljan iz mrežnega.



Slika 3.3: Primer enostavnega grafa

Graf je struktura[5], ki je sestavljena iz vozlišč in povezav. Definiran je v enačbi 3.9, kjer so:

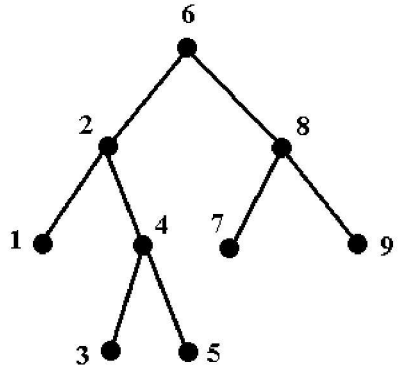
- V – končna množica vozlišč,
- E – končna množica povezav,
- θ – funkcija, ki preslikuje elemente iz E v elemente množice $((VxV) \bigcup V$

$$G = (V, E, \theta) \tag{3.9}$$

Graf lahko vsebuje cikle. Če povezavam dodamo še smer, dobimo usmerjen graf. Povezava med vozlišči A in B je usmerjena iz A proti B, če po njej lahko pridemo iz A v B, ne moremo pa priti iz B v A. Pot med vozliščema A in B je zaporedje povezav iz E, po katerem lahko pridemo iz vozlišča A v vozlišče B. V sliki 3.3 je primer grafa, kjer so $V = \{A, B, C, D\}$, $E = \{a, b, c, d, e, f, g\}$ in $\theta = \begin{pmatrix} a & b & c & d & e & f & g \\ \{A,B\} & \{A,B\} & \{A,C\} & \{B,C\} & \{B,C\} & \{B,C\} & \{B,D\} \end{pmatrix}$.

3.1.3 Hierarhični/drevesni model

Drevesni model je poenostavljen mrežni model[4]. V drevesnem modelu imamo eno začetno vozlišče, iz katerega gredo povezave do njegovih naslednikov, iz naslednikov v njihove naslednike in tako naprej. Ta model ne vsebuje ciklov in povezave obstajajo samo med vozliščem in njegovimi nasledniki.



Slika 3.4: Enostavno drevo

Predem lahko podamo formalno definicijo drevesa, moramo uvesti še pojem povezanega grafa. Graf je povezan, če za vsak par vozlišč iz V obstaja množica povezav iz E, po kateri lahko pridemo iz enega vozlišča do drugega[5].

Formalni ekvivalentni definiciji drevesa:

1. drevo je povezan graf brez ciklov,
2. graf je drevo, če:
   - nima ciklov,
   - za vsak par vozlišč obstaja natanko ena pot med njima,
   - če odstranimo eno (ni važno katero), dobimo nepovezan graf,
   - graf ima eno vozlišče več, kot je povezav.

V sliki 3.4 je podano enostavno drevo.

V mrežnem in drevesnem modelu so podatki shranjeni v vozliščih, odnose med posameznimi entitetami pa predstavljajo povezave.

Izkušen bralec bo upravičeno opazil podobnosti med drevesnim modelom in imeniškimi strukturami, kot so LDAP, JNDI, datotečni sistemi itd. Naštete imeniške strukture niso nič drugega kot implementacija drevesnega modela za specifične potrebe. V naslednjem poglavju bomo opisali specifikacijo drevesne podatkovne baze, ki jo uporablja Magnolia.

3.1.4 JCR

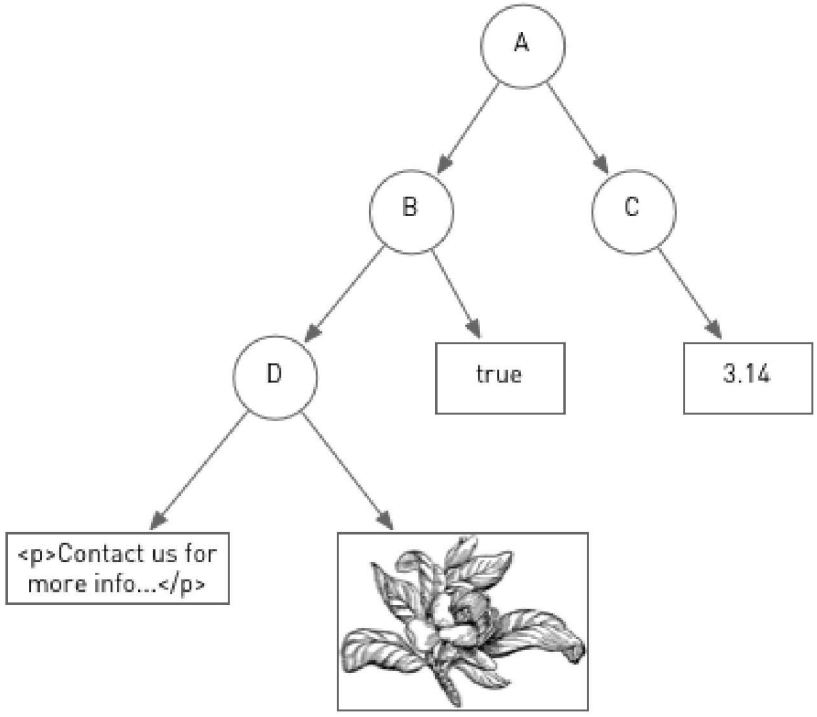
JCR je Javina specifikacija za drevesni model[6]. Celotna podatkovna struktura se imenuje repozitorij, razdeljena pa je na delovne prostore. V vsakem delovnem prostoru je shranjeno eno drevo. Povezave v repozitoriju predstavljajo potomstvo oziroma prednike. Obstajata dve vrsti entitet: vozlišča in lastnosti. Vozlišča so entitete, ki lahko vsebujejo lastnosti in ostala vozlišča – lahko imajo izhodne povezave. Lastnosti – atributi so entitete, ki ne morejo imeti izhodnih povezav – lahko so samo listi drevesa. Podatki se običajno shranijo v lastnosti, medtem ko vozlišča služijo za predstavitev relacij med entitetami. JCR je v jedru drevesni model, lahko pa simulira tudi mrežnega. To naredimo tako, da v lastnost nekega vozlišča shranimo pot, lahko relativno, lahko absolutno, do nekega drugega vozlišča. Začetne absolutne poti so vedno korensko vozlišče trenutnega delovnega prostora. Ob poizvedbah v bazo se bo namesto dejanskega vozlišča upoštevalo referencirano.

Vsak delovni prostor ima natanko eno začetno/korensko vozlišče, iz katerega se potem razveji celotna podatkovna struktura. Entitete so poimenovane s poljubnimi objekti tipa »string«, le korensko vozlišče je vedno prazen »string«. Enostaven primer je prikazan v sliki 3.5.

Lokacijo entitet podamo s potjo. Pot je po navadi zaporedje imen vozlišč, ločenih z znakom /. Poleg unikatne poti ima vsaka entiteta tudi identifikator. Identifikator je unikaten za vsako vozlišče znotraj delovnega prostora. To nam omogoča, da lahko dostopamo neposredno do entitete, brez da bi poznali njeno lokacijo v drevesu. Identifikator se ob premikanju entitete ne spremeni, zato je uporaben tudi za dostop do entitet, ki se veliko premikajo.

Dejanski podatki se shranjujejo v lastnosti. Kateri podatek je v lastnostih, lahko vidimo iz njihovega tipa. Možni tipi so vnaprej določeni[6] in so:

- »STRING«: shranjuje instance »java.lang.String«,
- »URI«: shranjuje instance tipa »java.lang.String«, ki se držijo URI-sintakse [RFC 3986],



Slika 3.5: Shematski prikaz »JCR delovnega prostora«[2]

- »BOOLEAN«: shranjuje primitive tipa »boolean«,
- »LONG«: shranjuje primitive tipa »long«,
- »DOUBLE«: shranjuje primitive tipa »double«,
- »DECIMAL«: shranjuje instance tipa »java.math.BigDecimal«,
- »BINARY«: shranjuje instance tipa »javax.jcr.Binary«,
- »DATE«: shranjuje instance tipa »java.util.Calendar«,
- »NAME«: shranjuje JCR-imena lastnosti. Iz uporabnikovega stališča so to instance tipa »java.lang.String«,
- »PATH«: shranjuje JCR-pot po drevesu. Iz uporabnikovega stališča so to instance tipa »java.lang.String«,
- »WEAKREFERENCE«: shranjuje kazalce na vozlišča. Ne preverja, ali vozlišče tudi dejansko obstaja,
- »REFERENCE«: shranjuje reference na vozlišča. Za razliko od »WEAKREFERENCE« lahko sem shranimo samo reference na obstoječa vozlišča.

Tukaj bi radi izpostavili razred »javax.jacr.Binary«, ki ga ni zaslediti v navadni Javi. To je JCR-razred, ki predstavlja binarne datoteke. JCR že vsebuje API za tvorbo teh objektov iz instanc »java.io.InputStream«.

Obstajata dve splošni vrsti lastnosti: lastnosti, ki shranijo en podatek, in lastnosti, ki shranijo več podatkov istega tipa. Lastnosti, ki shranjujejo en podatek, morajo imeti vrednost. Null ni veljavna vrednost. Lastnosti, ki shranjujejo več vrednosti, lahko imajo nič ali več vrednosti.

Tudi vozliščem lahko določimo različne tipe. Tip vozlišča določa njegovo strukturo. Tako lahko vsakemu vozlišču določimo, koliko naslednikov lahko ima, katere lastnosti mora vsebovati, katere niso obvezne itd. Tip »nt:base« je edini tip, ki ga določa specifikacija in vozlišč v ničemer ne

omejuje. Če ne določimo drugače, ga samodejno vsebujejo vsa vozlišča[6]. Uporaba lastnih tipov vozlišč ni obvezna, ampak je stvar odločitve pri arhitekturi posamezne podatkovne baze. Uporabnik do JCR-vsebin dostopa preko sej. Uporabnik je tukaj mišljen v najširšem možnem pomenu. Lahko je človek, nek zunanji proces itd. Za dostop do seje se mora prijaviti z vnaprej določenimi prijavnimi podatki, ki ga enolično določijo. Običajno sta to uporabniško ime in geslo.

Na voljo je več implementacij JCR-specifikacije. Magnolia CMS uporablja Apache Jackrabbit. Zaradi nekaj slabših razvojnih odločitev v preteklosti ne moremo prosto zamenjati implementacije JCR-ja, brez večjih posegov v kodo aplikacije. Razvijalci aktivno delajo na tej težavi, trenutno pa še niso odpravili težav.

3.1.5 Primerjava relacijskega in drevesnega modela

Relacijski in drevesni model sta tako različna, da ju je težko primerjati. Oba namreč dobro služita osnovnima nalogama podatkovne baze: shranjevanje in branje podatkov.

Najlažje ju primerjamo z vidika spreminjanja strukture. V relacijskem modelu moramo imeti vse entitete že vnaprej definirane. Vsako naknadno spreminjanje entitet in povezav med njimi zahteva dobro premišljen poseg v strukturo podatkovne baze. Poseg mora opraviti vzdrževalec podatkovne baze, razvijalci pa morajo potem še ustrezno prilagoditi aplikacijo. Pri aplikaciji, ki uporablja drevesno podatkovno bazo, programer popravi aplikacijo in baza se ji samodejno prilagodi[4]. Pri takih posegih moramo biti zelo previdni, saj lahko tak način spreminjanja podatkovne baze privede do slabo dokumentiranega in nejasnega podatkovnega modela, zato pri uporabi drevesnih podatkovnih baz potrebujemo izkušenejše razvijalce.

Zelo pomemben podatek je razširjenost. Relacijske baze so veliko bolj razširjene, kar pomeni veliko več dokumentacije in tudi razvijalcev. Če se odločimo za razvoj aplikacije z drevesno bazo, se moramo zanašati predvsem na svoje znanje in običajno plačljivo podporo.

Kot smo videli, sta drevesni in relacijski model tehnično enakovredna, vendar se moramo pri izbiri drevesnega modela zavedati praktičnih posledic.

3.2 Pregled glavnih lastnosti Magnoli-e

3.2.1 Razvoj spletnih strani

Pri razvoju spletnih strani moramo poznati naslednje pojme[2]:

- predloga
- območje
- komponenta
- obrazec

Vsaki spletni strani pripada ena predloga (slika 3.6). Predloga predstavlja neko zaokroženo logično in strukturno celoto, ki določi, kako bo stran videti. Vsaka predloga je sestavljena iz enega ali več območij.

Območje je del strani, ki združuje vsebinsko ali strukturno povezane elemente. Območja lahko vsebujejo druga območja ali komponente.

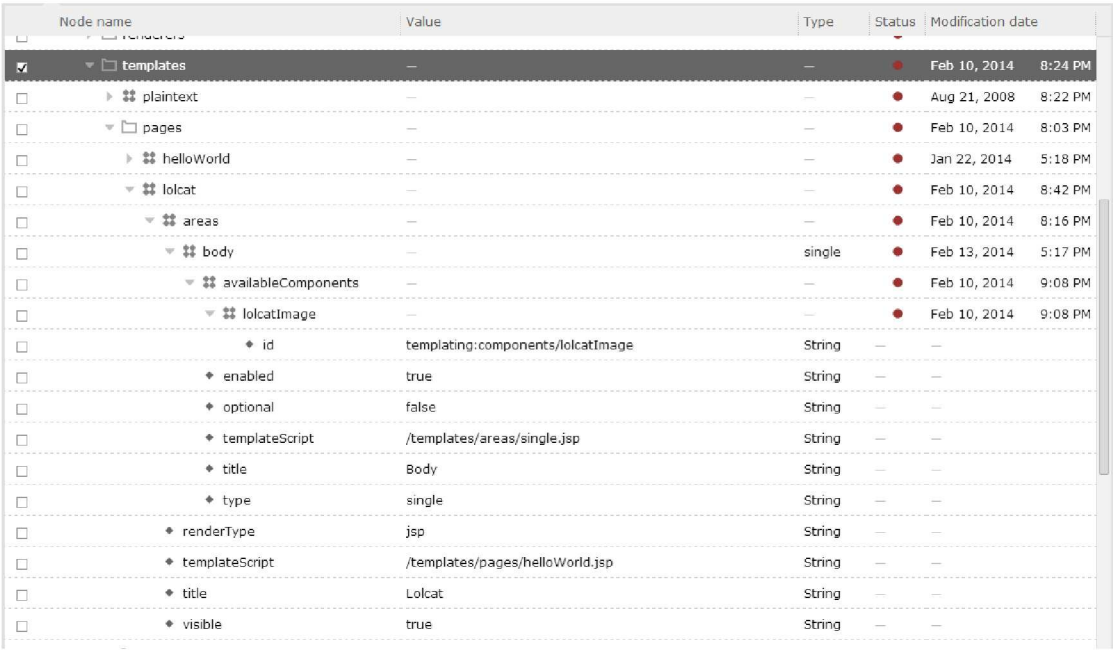
Komponente so atomarni del spletne strani, ki služi za prikaz točno določenega logičnega elementa.

Obrazci (slika 3.7) niso del končne spletne strani, ampak služijo za vnos podatkov v komponente ali predloge. Vsaki predlogi in komponenti lahko določimo javanski razred, ki vsebuje poslovno logiko.

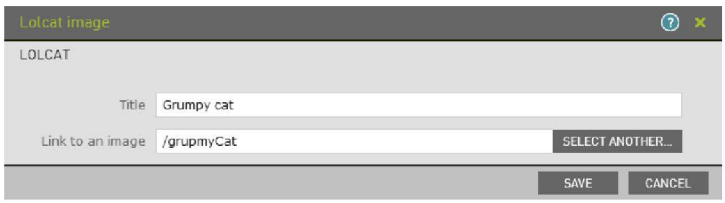
Celotna struktura spletnega mesta se shrani – preslika v JCR podatkovno bazo. Na sliki 3.8 je primer strani, ki jo lahko ustvarimo iz entitet na slikah 3.6 in 3.7. Na sliki 3.9 je stanje v bazi, ko stran ustvarimo.

3.2.2 Administrativni vmesnik

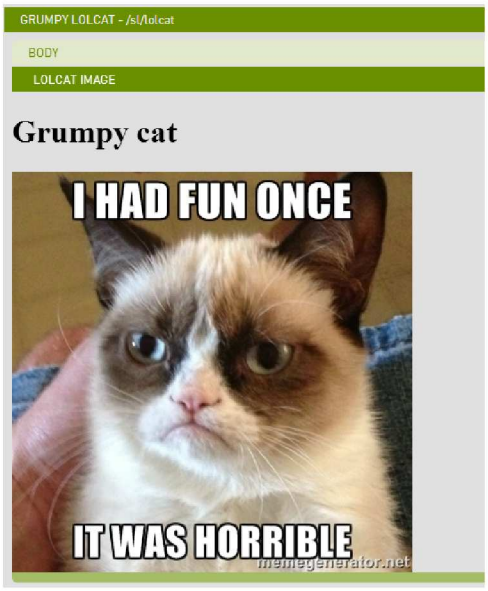
Vsi CMS-i imajo vmesnik za upravljanje z vsebinami in Magnolia ni nobena izjema. Sestavljen je iz več različnih komponent, ki jih Magnolia imenuje



Slika 3.6: Primer predloge »lolcat« z enim območjem (»body«), ki lahko vsebuje eno komponento tipa »lolcatImage«.



Slika 3.7: Obrazec za vnos podatkov v komponento »lolcatImage«



Slika 3.8: Spletna stran v Magnoliinem urejevalniku

Node name	Value	Type	Status	Modification date
title	–	–		Feb 10, 2014 8:15 PM
sl	–	–		Feb 13, 2014 5:17 PM
lolcat	–	–		Feb 13, 2014 5:18 PM
body	–	–		–
0	–	–		Feb 13, 2014 5:18 PM
image	/grupmyCat	String	–	–
title	Grumpy cat	String	–	–
title	Grumpy lolcat	String	–	–
title	lang root	String	–	–

Slika 3.9: Stanje JCR-baze, kjer je shranjena spletna stran

aplikacije[2]. Ker Magnolia vse shranjuje v drevesno podatkovno bazo, aplikacije niso nič drugega kot kontekstualen pogled na dele drevesne strukture. Pogled je kontekstualen v smislu, da prikaže in omogoča urejanje samo tistih vozlišč in lastnosti, ki so smiselna za namen aplikacije. Če na primer potrebujemo aplikacijo, ki bo omogočala urejanje samo naslovov spletnih strani, potem jo bomo omejili na delovni prostor »website«, kjer bo lahko prikazala vsa vozlišča (vsako vozlišče predstavlja spletno stran), ampak samo lastnosti »title«.

V sliki 3.9 je prikazana aplikacija za surovo prikazovanje in urejanje podatkov v JCR delovnem prostoru. Če bi taka aplikacija iz slike 3.9 prikazovala samo lastnosti »title«, bi dobili aplikacijo, ki je bila opisana v prejšnjem odstavku. V poglavju 4 bomo tudi razvili lastno aplikacijo za vnos vsebin, ki bo zelo prilagojena potrebam modula.

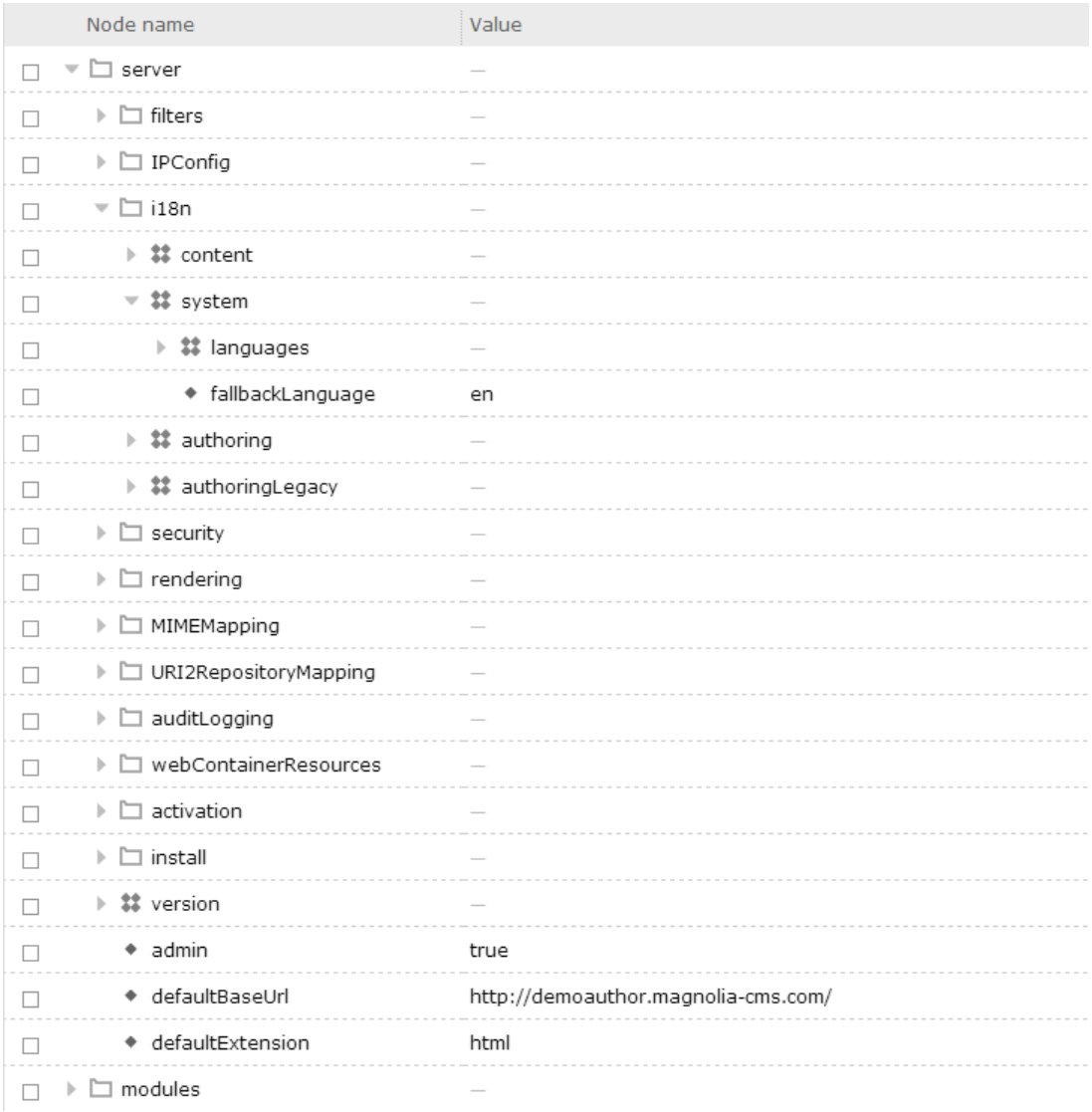
3.2.3 Konfiguracija

Magnolia ima izredno modularno zastavljeno konfiguracijo. Ker se bo naše rešitve nastavljalo na identičen način, bomo tu za lažje razumevanje najprej opisali splošne koncepte in načine nastavljanje posameznih sklopov osnovne nastavitve Magnolie.

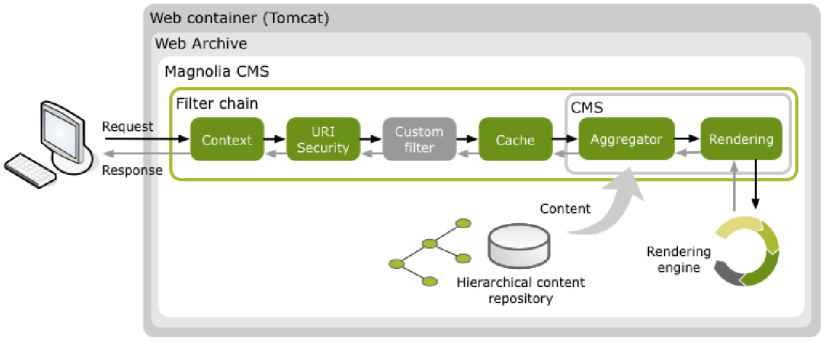
Konfiguracija je razdeljena med dva večja segmenta, »server« in »modules« (slika 3.10). Celotna konfiguracija je shranjena v JCR (poglavje 3.1.4) in oba segmenta nista nič drugega kot dve vozlišči. To omogoča prosto spreminjanje in razširjanje konfiguracije preko Magnoliinega vmesnika za upravljanje z vsebinami v JCR podatkovni bazi.

Segment »server« vsebuje konfiguracijo, ki vpliva na celotno aplikacijo. Spodaj so našteta vozlišča v tem segmentu in podan kratek opis funkcionalnosti.

Filter Filtri v Magnolii so procesorji, ki obdelujejo HTTP-zahtevo. Izvršujejo se v takem vrsten redu, kot so shranjeni v JCR[2]. Vsako vozlišče mora imeti vsaj dve lastnosti: »class« in »enabled«. Lastnost class predstavlja javan-



Slika 3.10: Pogled na konfiguracijo segmenta server iz Magnolijinega administrativnega vmesnika.



Slika 3.11: Shematski prikaz potovanja HTTP-zahtev in HTTP-odgovorov skozi verigo filtrov[2].

ski razred, ki opravlja procesiranje. Razred mora implementirati vmesnik »info.magnolia.cms.filters.MgnlFilter«. Lastnost »enabled« nam pove, ali je filter trenutno vklopljen. Če je nastavljen na »false«, se filter ne izvede. JCR–vozlišče filtra lahko vsebuje poljubno poddrevo, ki natančneje nastavlja filter, če implementacija razreda v lastnosti »class« to podpira, drugače so dodatni podatki samo balast. Shematski prikaz potovanja HTTP-zahtev in HTTP-odgovorov skozi verigo filtrov je podan v sliki 3.11.

IPConfig V tem vozlišču lahko določimo pravila za filtriranje zahtev po IP-naslovih. Dovolimo lahko, ali blokiramo zahteve iz poljubnih IP-naslovov.

I18n To vozlišče določi podprte jezike HTTP-zahteve in mehanizem za detekcijo jezika. Magnolia potem samodejno lokalizira vsebino, če naša aplikacija podpira jezik. Drugače nastavi jezik na privzeti jezik, ki ga lahko tudi poljubno nastavimo. Magnolia lokalizacijo deli na tri dele:

- »system«: jezik administracijskega vmesnika,
- »authoring«: pri vnosu podatkov preko obrazcev se v obrazcu pojavi izbirni meni za jezik,
- »content«: Magnolia za vsak jezik ustvari ločeno JCR-drevo za vsebine,

Security V tem vozlišču je nastavljen celotni varnosti mehanizem Magnolie (poglavje 3.2.4)

Rendreing Rendering inicializira celoten sistem za izris vsebin.

MIMEMapping V tem vozlišči so shranjene vse podprte MIME-preslikave in njihove HTTP-glave.

URI2RepositoryMapping Ta filter na podlagi URI-ja določi, iz katerega repozitorija se bo servirala vsebina.

AuditLogging Tukaj so shranjene nastavitve za beleženje operacij nad JCR-repozitorijem.

WebContainerResources V tem vozlišči nastavimo, katero vsebino bo serviral aplikacijski strežnik sam, ne pa naša aplikacija.

Activation Tukaj nastavimo, kam se bo aktivirala vsebina, če je to avtorska instanca.

Vozlišče »server« vsebuje še tri dodatne lastnosti:

- »admin«: pove, ali je to javna ali avtorska instanca,
- »defultBaseUrl«: del URL-ja, ki se predpne vsem polnim absolutnim HTTP-povezavam (linkom),Za to nalogo smo razvili boljši mehanizem, zato naša instanca Magnolie ne podpira te lastnosti,
- »defaultExtension«: katero končnico bo Magnolia pripenjala samodejno ustvarjenim URL- povezavam.

V segmentu »modules« so shranjene vse nastavitve dodatnih modulov. Vsak modul ima svoj izbor nastavitev. Nekaterih modulov se ne da nastavljati in imajo samo zapisano različico, nekateri pa so tudi kompleksnejši od vozlišča »server«.

3.2.4 Varnost v Magnoliji

Magnolia pozna štiri varnostne entitete[2]:

- uporabniki: ljudje, ki upravljajo z vsebino,
- sistemski uporabniki: ljudje, ki lahko konfigurirajo Magnolio,
- skupine: množice uporabnikov s podobni pravicami,
- vloge: v vlogah so shranjene dejanske nastavitve ACL (opis spodaj).

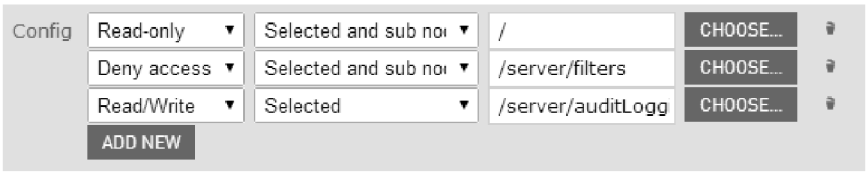
ACL ACL ali Access control list je dejanski mehanizem za omejevanje dostopa v JCR-bazi. Ker je JCR-baza drevesne oblike, lahko za vsako vozlišče in njegove naslednike določimo pravice glede na vlogo. Obstajajo tri vrste dostopov:

- »Read/Write«: vloga s tem dostop lahko bere in spreminja podatke v vozlišču,
- »Read-only«: vloga s tem dostopom lahko same bere podatke iz vozlišča,
- »Deny access«: vozlišče za to vlogo ne obstaja.

Ker tak način omejevanja včasih ni dovolj, lahko omejujemo dostop do tudi do posameznih URL-jev. Pri filtriranju lahko uporabimo regularne izraze. Ko imamo jasno definirane vloge, jih lahko združujemo v skupine, ki jim dodajamo uporabnike. Vlogo lahko dodamo tudi neposredno uporabniku. V sliki 3.12 je primer enostavne vloge.

3.3 Lastni moduli

Ker nobeno ogrodje ne more vnaprej predvideti vseh potreb končnega uporabnika, imajo praktično vsa ogrodja možnost razvijanja lastnih funkcionalnosti. Magnolia tukaj ni nobena izjema. Lastne rešitve za Magnolio se



Slika 3.12: Primer enostavne vloge. Uporabnik s to vlogo vidi celoten »config« delovni prostor, razen »/server/filters«. Ureja pa lahko samo nastavitve »AuditLogging«.

imenujejo moduli. Magnolia ima predpisan standardni način razvijanja modulov, ki ga bomo tudi na kratko predstavili.

3.3.1 Razvoj modula v Magnoliji

Če uporabljamo orodje Maven, lahko vsak projekt ustvarimo iz ukazne vrstice[1]. Maven nam potem samodejno ustvari pravilno strukturo v datotečnem sistemu, ki jo lahko odpremo v poljubnem IDE-ju. Pri konfiguraciji modulov imamo na voljo JCR-vozlišča, podana v tabeli 3.4. Nobeno vozlišče ni obvezno. Z naštetimi vozlišči lahko določimo obnašanje modula v aplikaciji. Vozlišča »commands«, »dialogs«, »apps« in »messageViews« služijo za ustvarjanje aplikacij v administrativnem vmesniku (več v poglavju 4). V »templates« lahko zapišemo definicije predlog za HTTP-strani (poglavje 3.2.1). V vozlišče »config« oziroma poljubno vozlišče, ki ni definirano v tabeli, lahko zapišemo nastavitve, ki so povezane s poslovno logiko modula. Če bi želeli v administracijskem vmesniku narediti imenik zaposlenih v nekem podjetju, bi s pomočjo vozlišč »commands«, »dialogs«, »apps« ustvarili uporabniški vmesnik za vpis, prikaz in posodabljanje zaposlenih. Ustvarili bi tudi novo vozlišče »zaposleni«, kamor bi zapisali poljubno drevesno strukturo, ki bi predstavljala zaposlene v podjetju. Poslovna logika tega modula bi omogočala naštete operacije nad zaposlenimi in bi jo lahko uporabili kjer koli v aplikaciji.

[1]Ukaz je »mvn archetype:generate -DarchetypeCatalog=http://nexus.magnolia-cms.com/content/groups/public/«.

vozlišče	namen	podrobnosti
commands	Definira ukaze	Množica vozlišč, ki konfigurira možne akcije v modulu.
config	Specifična konfiguracija posameznega modula	Vsebuje parametre in podatke vezane na modul.
dialogs	Obrazci	Vsebuje konfiguracijo obrazcev za vnos vsebine, ki jih modul doda v aplikacijo.
fieldTypes	Definicije vrst vozlišč	Tukaj lahko definiramo nove tipe JCR-vozlišč
renderers	Tukaj lahko nastavimo logiko za izris spletnih vsebin	Magnolia privzeto uporablja FreeMarker in JSP-jezik za predloge. Tukaj lahko definiramo nove jezike ali spremenimo obnašanje obstoječih.
templates	Konfiguracija predlog	Tukaj lahko konfiguriramo predloge, območja in komponente
virtualURI-Mapping	Pravila za izris vsebin glede na URI	Tukaj lahko nastavljamo pravila, kako se bodo določeni HTTP-zahtevki izrisali glede na njihov URI.
messageViews	Definira vsebino sporočil v administrativnem vmesniku	Tukaj lahko dodatno katere akcije in na kakšen način bodo vračale informacijo uporabniku v administrativnem vmesniku.
apps	Definira aplikacije	Množica vozlišč, ki definira aplikacijo

Tabela 3.4: Glavna vozlišča za nastavljanje modulov.

Ker moduli običajno potrebujejo tudi začetno stanje, imamo poleg konfiguracije v JCR-repozitoriju na voljo še opisno datoteko. Nahaja se v imeniku »META-INF/magnolia« in je del kode projekta. Opisna datoteka je formata XML in vsebuje elemente, podane v tabeli 3.5. Obvezna elementa sta samo »version« in »name«.

Za lažjo predstavo je v sliki 3.1 koda enostavnega deskriptorja. Razvidno je, da bo modul ustvaril JCR-vozlišče »form« (vozlišče name v datoteki). Ker nima podanega repozitorija, se bo celotna konfiguracija zapisala v repozitorij config, kar je privzeta vrednost. V atribute razreda, podanega v vozlišču »class«, se bodo samodejno zapisale vrednosti iz JCR vozlišča config. Avtor modula in upravljavec vozlišča morata sama poskrbeti, da se bodo tipi atributov v razredu ujemali s tipi lastnosti v JCR-vozlišču, drugače modul ne bo deloval. V vozlišču »versionHandler« je podan razred, ki trenutni različici modula pravilno prilagodi vrednosti v JCR- drevesu. Če smo na primer v aplikaciji najprej uporabili različico modula 1.4.3, je ta razred samodejno ustvaril začetno stanje. Predpostavljamo, da je ta različica vsebovala JCR-vozlišče po imenu poSts. Razvijalci modula so napako odkrili in v različici 1.4.4 vozlišče preimenovali v »posts«. Če bi aplikacijo posodobili z različico 1.4.4, potem aplikacija ne bi delovala. Zato ima Magnolija vgrajeno knjižnico, s katero lahko povemo, da ob prehodu med različicami 1.4.3 in 1.4.4 JCR-vozlišče »poSts« preimenujemo v »posts«. V vozlišče »dependencies« zapišemo module in njihove različice, ki jih mora Magnolia namestiti, preden namesti modul »form«.

Ostale lastnosti in mehanizme bomo opisali pri razvoju modulov, ko se bo za to pojavila potreba.

element	opis
name	Funkcionalno ime modula. To bo tudi ime glavnega nastavitvenega vozlišča v vozlišču »modules«.
displayName	Opisno ime modula. V glavnem se uporablja pri beleženju.
description	Poln opis modula. Namenjen je uporabniku.
class	Ime javanskega razreda, ki definira modul.
versionHandler	Ime javanskega razreda, ki popravlja JCR-nastavitve, ki so se spreminjale med razvojem modula.
version	Trenutna različica modula.
properties	Vrednosti, ki jih lahko uporabi »versionHandler«.
dependecies	Moduli, od katerih je ta modul odvisen. Ta lastnost pove, kateri moduli morajo biti nameščeni, preden se lahko namesti trenutni modul.
servlets	Nastavitve za »servlets«, ki jih vpelje trenutni modul. Ob prvi in samo prvi namestitvi modula se bodo samodejno nastavile v »servlets filtru«.
repositories	JCR-repozitoriji, ki jih bo uporabljal modul. Po potrebi jih postopek namestitve ustvari sam.
components	Javanski razredi, ki jih vsebuje modul. Razrede, ki so definirani tukaj, bo Magnolia naredila dostopne celotni aplikaciji.

Tabela 3.5: Elementi opisne datoteke

Algoritem 3.1 Opisna datotka modula form, ki je del standardne namestitve Magnolije.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module SYSTEM "module.dtd" >
<module>
        <name>form</name>
        <displayName>Magnolia Form Module</displayName>
    <description></description>
        <class>info.magnolia.module.form.FormModule</class>
    <versionHandler>
          info.magnolia.module.form.setup.FormModuleVersionHandler
    </versionHandler>
    <version>1.4.4</version>
        <dependencies>
                <dependency>
                        <name>core</name>
                        <version>4.5.1/*</version>
                </dependency>
                <dependency>
                        <name>magnolia-4-5-migration</name>
                        <version>1.2/*</version>
                </dependency>
                <dependency>
                        <name>templating</name>
                        <version>4.5.1/*</version>
                </dependency>
                <dependency>
                        <name>mail</name>
                        <version>4.5.1/*</version>
                </dependency>
        </dependencies>
</module>
```

Poglavje 4

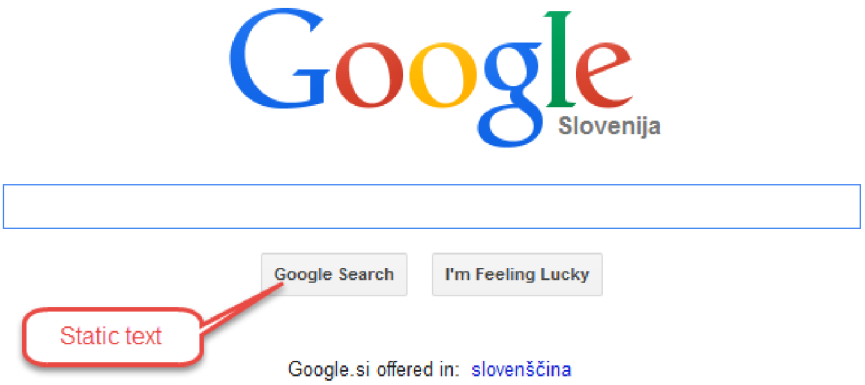
Modul za lokalizacijo statičnih vsebin

Večino vsebin na spletne strani vnašajo ljudje, obstajajo pa vsebine, ki so statične. To pomeni, da je na istem delu spletne strani vedno isto besedilo. Če si predstavljamo neko spletno stran, ki ima v glavi možnost iskanja, iskanje vedno sprožimo s klikom na gumb išči. Glava je enaka za celo spletno mesto, ne glede na to, kje na njem se nahajamo. V takih primerih uporabniku nima smisla dodajati dodatne odgovornosti, ampak lahko to besedilo naredimo statično. To pomeni, da ga uporabnik ne more spreminjati. Težava se pojavi pri lokalizaciji takih besedil. Primer smo podali v sliki4.1.

4.1 Klasični načini lokalizacije

Pri razvoju večjezičnih rešitev se bo vsakdo prej ali slej srečal s pojmoma i18n in l10n. I18n (angl. *internationalization*) je postopek načrtovanja in razvoja rešitve, ki podpira večjezičnost. L10n (angl. *localization*) je dodajanje podpore jezika v rešitev, ki je bila razvita z večjezičnostjo v mislih (i18n).

Če razvijamo spletno mesto za prodajalca, ki trenutno deluje le v Sloveniji, in vemo, da se bo razširil tudi v druge države, potem načrtujemo aplikacijo po principu i18n. Ko se prodajalec razširi tudi na Hrvaško, mo-



Slika 4.1: Googlova domača stran z označenim primerom statičnega besedila

ramo v spletno mesto dodati podporo tudi Hrvaščini – postopek L10n.

Če imamo na slovenskem spletnem mestu besedo »dom«, potem se mora na hrvaškem spletnem mestu izpisati »kuća«. Ker imamo dve različni besedi, ju moramo skupaj povezati s ključem, recimo »home«. Potrebujemo še metodo, ki bi bila sposobna pridobiti pravo besedilo: *prevedi (jezik, ključ)*. Če bi tako metodo sprožili kot »*prevedi (slovenščina, home)*«, bi kot rezultat dobili dom. Če bi uporabili hrvaščino, torej »*prevedi (hrvaščina, home)*«, bi dobili »kuća«.

Lokalizacija v splošnem vsebuje tri entitete: jezik, ključ in prevod. Jezik (angl. *locale*) je običajno oblike »dr_JZ«, kjer je »dr« koda države, »JZ« pa koda jezika, ki se uporablja v tej državi. To ni edina možnost predstavitve jezika, več o možnih oblikah si bralec lahko prebere v specifikaciji standarda BCP 47[7].

Različne rešitve imajo lahko zelo različne implementacije postopka pridobivanja prevodov. Gettext in Java i18n jih bereta iz datotek, ogrodje CakePHP iz relacij v podatkovni bazi, ...

4.2 Rešitev v Magnoliji

Magnolia že vsebuje podporo za i18n, ki temelji na Java i18n. Java i18n je implementirana s pomočjo datotek, kar nam povzroča težavo, saj jih med

izvajanjem aplikacije ne moremo spreminjati. Če smo bolj natančni, spremenimo jih lahko, vendar se spremembe zaradi načina implementacije ne bodo osvežile v aplikacijo, dokler ponovno ne zaženemo celotnega aplikacijskega strežnika.

Razvijalci Magnolie se tega zavedajo, zato so dodali tudi možnost lokalizacije pri vnosu vsebin. Če se odločimo za tak pristop, lahko lokaliziramo samo celotno komponento ali spletno stran, ne pa samo posameznih delov znotraj komponent ali spletnih strani.

Ker hočemo takojšnje osveževanje sprememb in natančno lokalizacijo delov strani, smo bili primorani razviti lasten modul.

Pri razvoju te rešitve smo morali stalno delati z dvema omejitvama:

1. v predlogi za HTML mora biti ta vsebina statična. To pomeni, da bo ena predloga pokrila vse jezike,

2. uporabnik lahko to vsebino vnaša ločeno od dejanske spletne strani. Ustvariti moramo uporabniški vmesnik za vnos prevodov.

Konceptualno smo si rešitev zamisli na naslednji način:

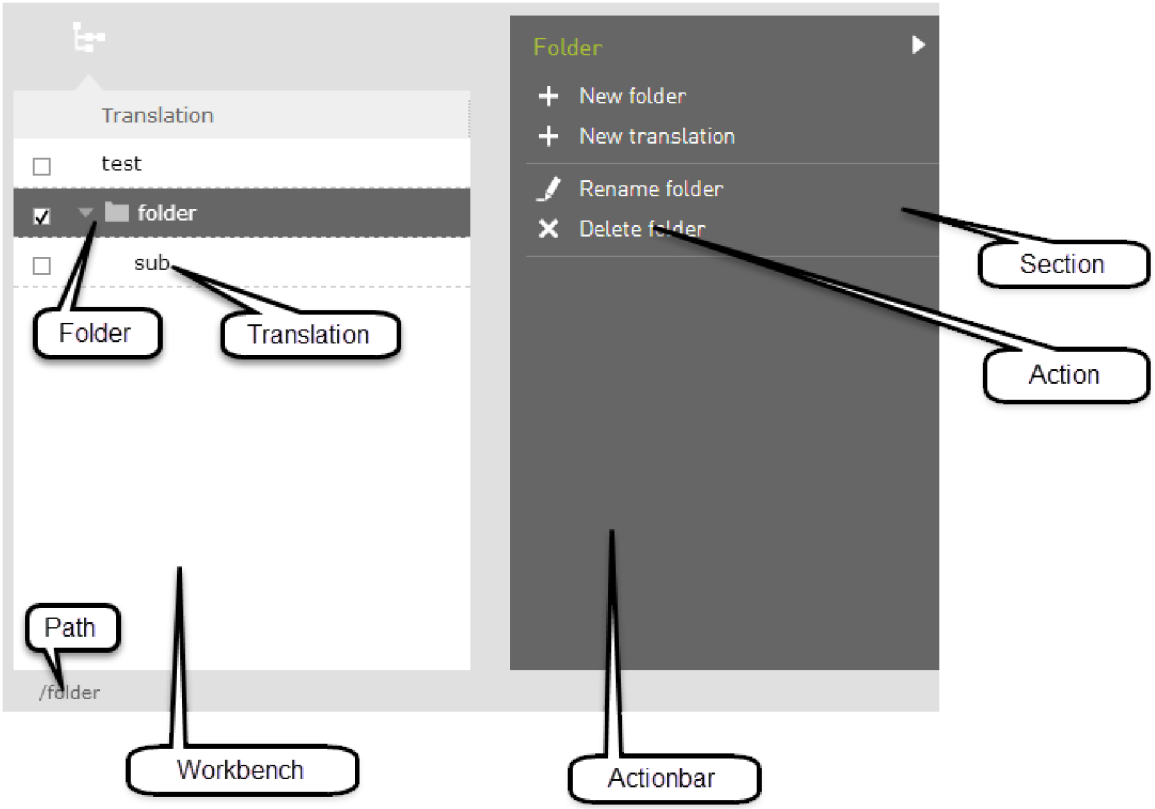
- vsi podatki se shranjujejo v JCR-bazo,

- HTML-datoteke do podatkov dostopajo preko unikatnega ključa,

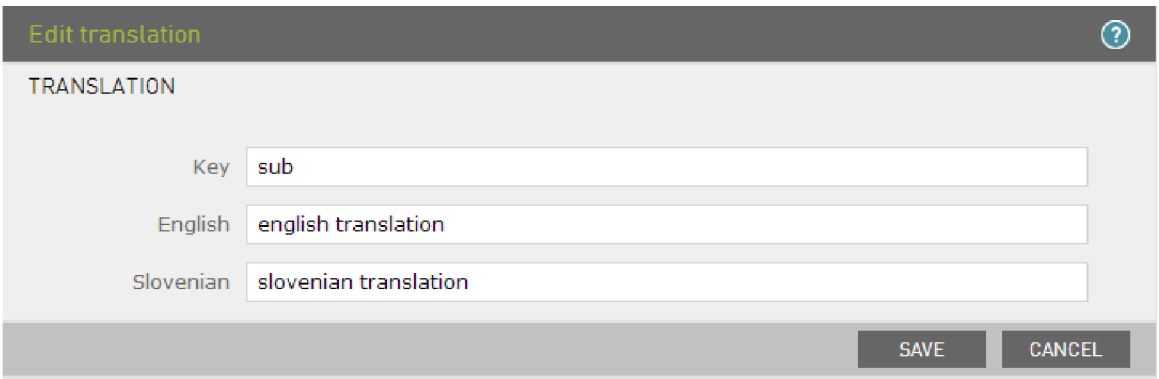
- če aplikacija poskuša dostopati do prevoda ključa, ki ne obstaja, aplikacija sama ustvari vse manjkajoče člene, za vrednost prevoda pa vzame prazen niz,

- uporabnik vnaša podatke preko administracijskega vmesnika.

Na slikah 4.2 in 4.3 je viden končnen vmesnik. Slika 4.2 vsebuje imena delov vmesnika. Natančneje so opisani v poglavju 4.4.



Slika 4.2: Brskalniška aplikacija z opisi posameznih delov



Slika 4.3: Obrazec za vnos prevoda

4.3 Razvoj zalednega dela

Najprej je bilo treba ustvariti ločen modul, ki smo ga poimenovali »pmgnl-utils«. Za modul smo ustvarili nov delovni prostor z imenom modula. V ta delovni prostor bomo shranjevali tudi vse podatke. Ker pa podatki o prevodih ne bodo edini podatki v tem delovnem prostoru, je treba že na začetku povedati, katero bo korensko vozlišče zanje. Izbrali smo vozlišče »/translations«.

To vozlišče bo za vozlišča imelo naslednike dveh tipov:

- imenik (angl. *directory*), koda tipa je »mgnl:folder«,
- prevod (angl. *translation*), koda tipa je »mgnl:translation«.

Imeniki so vozlišča brez lastnosti, služijo samo za združevanje povezanih prevodov. Imeniki lahko vsebujejo prevode in druge imenike. **Prevodi** so vozlišča brez potomcev. Imajo samo lastnosti, v katerih so shranjeni podatki. Vsebujejo naslednje lastnosti:

- ključ: ključ (i18n ime) je identifikator prevoda v tem vozlišču. Je enak za vse jezike, zato se bo HTTP-predloga sklicevala na vrednost te lastnosti. Lastnost je tipa NAME,
- jezik: vsako vozlišče vsebuje nič ali več teh lastnosti. Ime lastnosti je enako kodi jezika, vrednost lastnosti pa dejanski prevod. Lastnost je tipa STRING.

Tukaj bi radi še opozorili, da besedo prevod uporabljamo v širšem pomenu. Ni nujno, da predstavlja prevod nekega besedila. Lahko predstavlja tudi format zapisa časa, decimalno ločilo itd. Do posameznega zapisa v JCR-ju tako dostopamo s potjo do ustreznega vozlišča in z želenim jezikom. Primer poti: »/navigation/titles/main«. »Navigation« in »titles« sta v tem primeru imenika, medtem ko je »main« vozlišče tipa prevod. Vozlišče s prevodom je vedno na zadnjem mestu v poti. Ker za razvoj uporabljamo JSP-jezik za predloge, smo zanj razvili tudi javanski razred, ki to delo opravlja

samodejno. Poimenovali smo ga »Localize«, sprejme pa dva parametra: pot in jezik. Jezik je parameter in je opcijski, ker ga lahko dobimo samodejno, če smo pravilno nastavili vozlišče i18n v nastavitvah za strežnik. Potreben pa je, ker lahko imamo na strani vsebine, katerih jezik se ne spreminja. Tipičen primer so običajno meniji za izbor jezika strani.

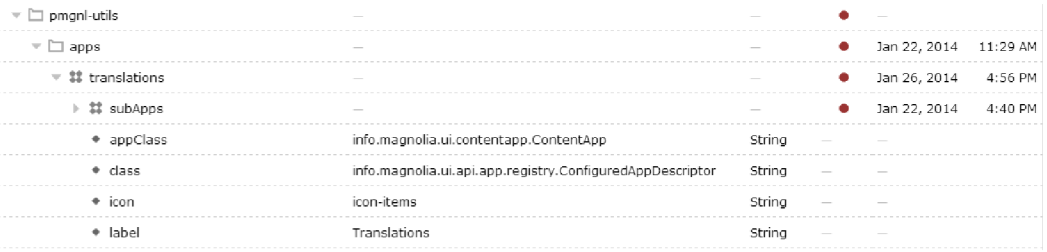
Če vse skupaj primerjamo s splošnimi načini lokalizacije, ugotovimo, da:

- entiteto jezik samodejno pridobimo iz Magnolie,
- je entiteta ključ v naši rešitvi pot do vozlišča tipa »prevod«,
- je entiteta »prevod« vrednost lastnosti za posamezen jezik v vozlišču tipa »prevod«,
- funkcijo iskanja pravilnega besedila (metoda prevedi iz poglavja 4.1) opravlja razred »Localize«.

4.4 Razvoj vmesnika za vnos vsebin

Vmesnik za vnos vsebin bo aplikacija v Magnoliinem administracijskem vmesniku. Bralcu toplo priporočamo, da si pomaga s slikami, ki so referencirane pri opisu posameznih sklopov, saj se bo sicer izgubil. V JCR-vozlišču modula ustvarimo imenik »apps« (slika 4.4), če le ta še ne obstaja. Korenska vozlišča v tem imeniku predstavljajo posamezne aplikacije tega modula. V tem primeru se vozlišče imenuje »translations«. Vozlišče vsebuje naslednje lastnosti:

- »appClass«: polno ime javanskega razreda, ki definira obnašanje/tip aplikacije,
- »class«: polno ime javanskega razreda, v katerega se preslika JCR-konfiguracija aplikacije,
- »icon«: ikona, ki se bo prikazala v izbirnem meniju za aplikacije,



Slika 4.4: Koren GUI-aplikacije

- »label«: opisno ime aplikacije. Končni uporabnik pozna aplikacijo pod tem imenom.

Poleg lastnosti aplikacije vsebuje še vozlišče »subApps«, v katerem so definirane še aplikacije, ki jih lahko poženemo iz glavne aplikacije. Aplikacija za vnos prevodov pozna dve podaplikaciji:

- »browser«: služi za strukturni prikaz JCR-drevesa in podatkov v njem,
- »detail«: služi za urejanje posameznih elementov v drevesu.

Aplikacija »browser« (slika 4.5) vsebuje dve pomembni lastnosti:

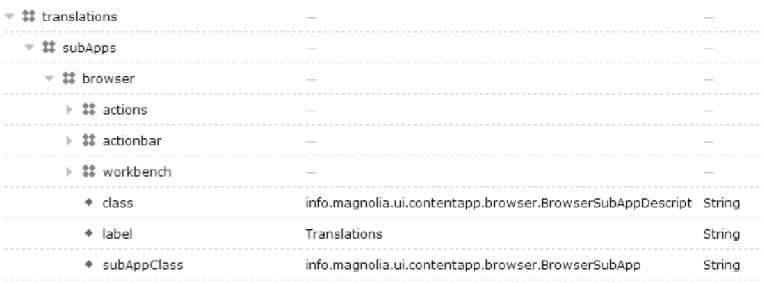
- »subAppClass«: definira obnašanje aplikacije,
- »class«: preslika konfiguracijo iz JCR-drevesa v javanski razred.

Vse brskalniške aplikacije so razdeljene na 3 dele:

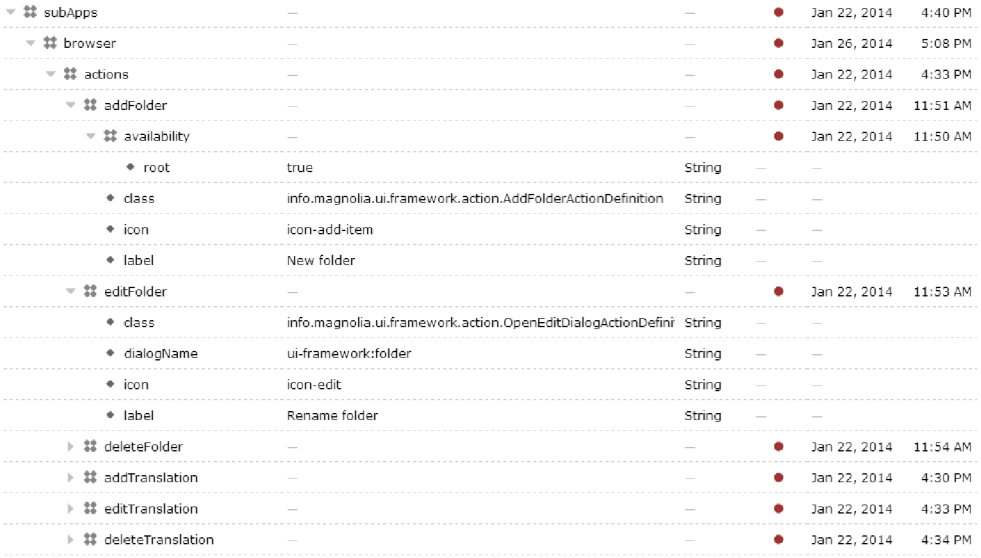
- »actions«: tukaj definiramo operacije, ki jih izvajamo nad elementi v JCR-ju,
- »actionbar«: orodna vrstica z akcijami,
- »workbench«: okno, ki služi za prikaz elementov.

Brskalniška aplikacija vsebuje šest operacij (slika 4.6): dodaj, zbriši, uredi imenik ter dodaj, uredi in izbriši prevod. Vsaka akcija vsebuje 3 lastnosti:

- »class«: javanski razred, ki definira obnašanje akcije,



Slika 4.5: Brskalniška aplikacija

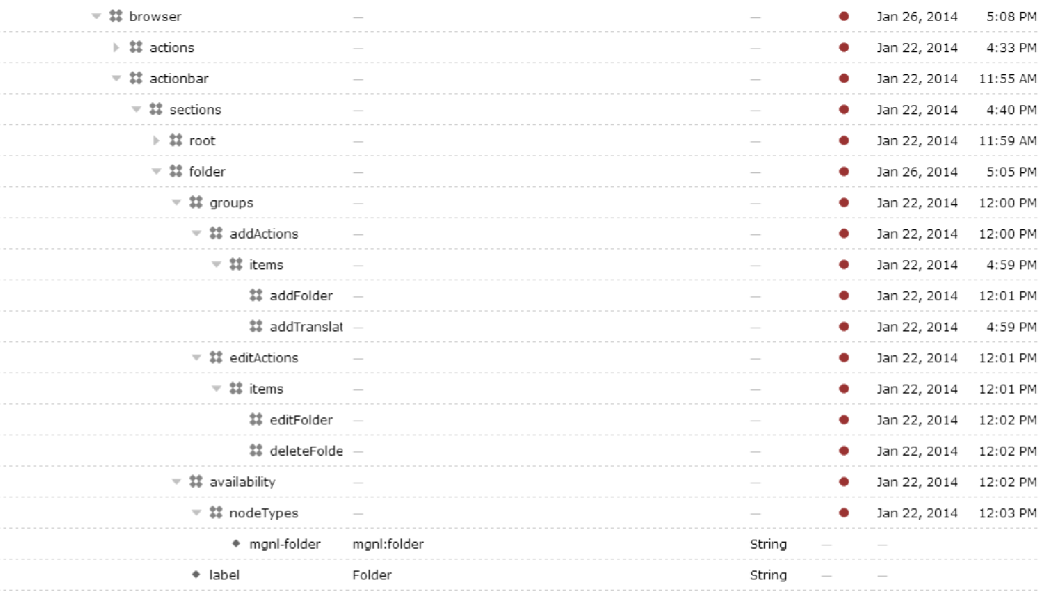


Slika 4.6: Operacije v brskalniški aplikaciji

- »icon«: ikona aplikacije v orodni vrstici,
- »label«: ime akcije v orodni vrstici.

Poleg tega lahko ima še vozlišče »availability«, s pomočjo katerega lahko natančneje določimo, kje bo akcija na voljo, ali bo na voljo v korenskem vozlišču ali samo na določenih tipih vozlišč itd.

Tudi orodno vrstico nastavimo preko JCR-ja (slika 4.7). Orodno vrstico razdelimo na sekcije. Vsaka sekcija ima svoja pravila, kdaj naj se izriše. Sekcije so razdeljene v skupine. V GUI-ju so skupine med seboj ločene z vertikalno črto. V vozlišču za skupine so potem prazna vozlišča, ki definirajo



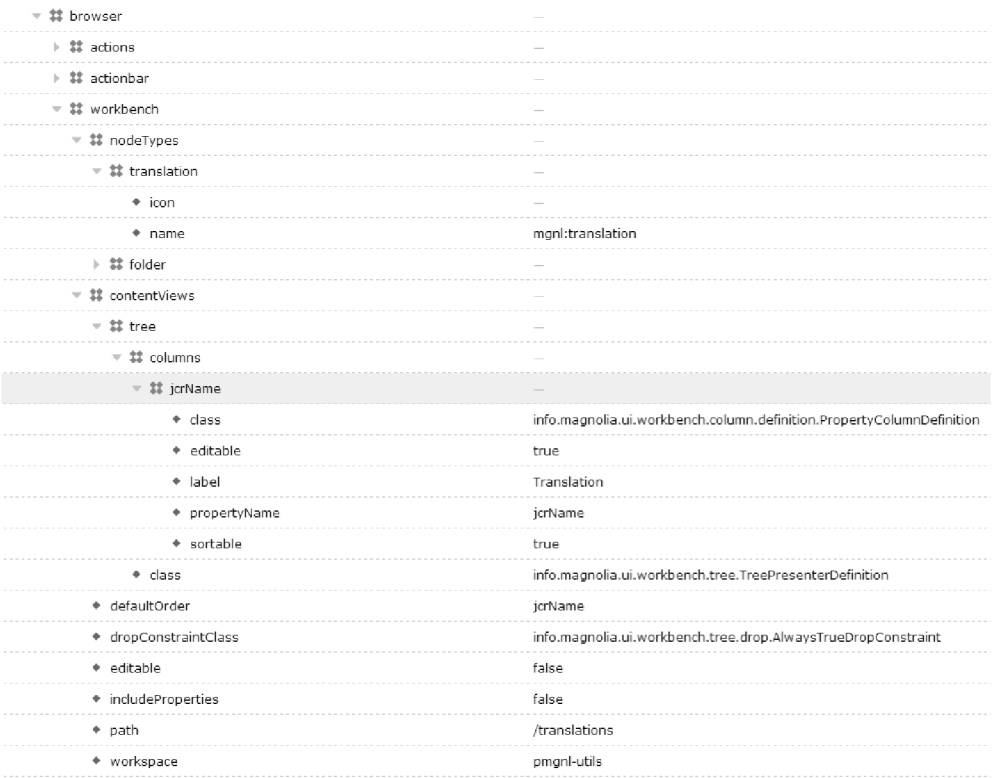
Slika 4.7: Orodna vrstica brskalniške aplikacije

akcije. Imeti morajo isto ime kot korensko vozlišče referencirane akcije.

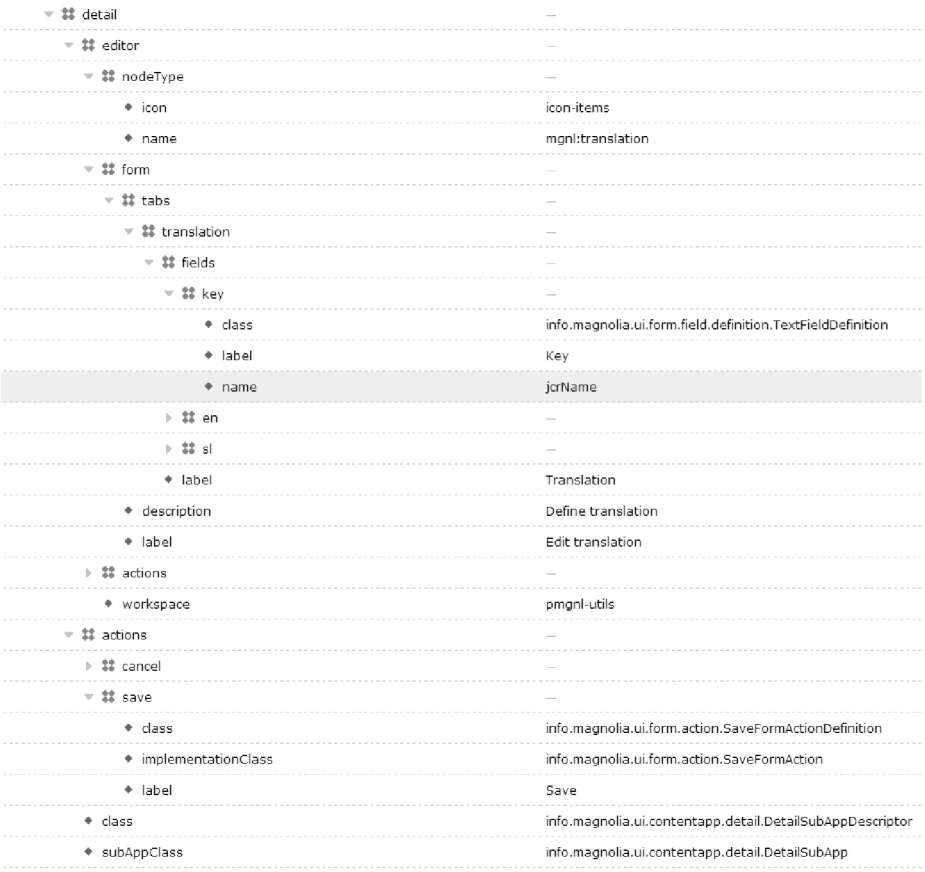
Zadnji del brskalniške aplikacije je vozlišče »workbench«(slika 4.8). V njem nastavimo glavno okno, kjer se prikazuje vsebina. Vsebuje pet lastnosti:

- »defaultOrder«: ime lastnosti, po kateri se bodo vozlišča sortirala,
- »dropConstraintClass«: ime javanskega razreda, ki določa pravila za mehanizem »vleci in spusti«,
- »editable«: ali lahko prikazane lastnosti urejamo kar znotraj samega okna,
- »path«: pot do vozlišča v delovnem prostoru. Izbrano vozlišče služi kot korensko vozlišče za prikazane vsebine,
- »workspace«: ime delovnega prostora, v katerem je vsebina.

Poleg splošnih lastnosti moramo v vozlišču »contentViews« definirati še različne poglede. Magnolia podpira drevesni pogled, seznam, predogled in



Slika 4.8: Nastavitve prikaza vsebin v brskalniški aplikaciji



Slika 4.9: Nastavitve aplikacije za vnos vsebin

iskalnik. Ker celotni mehanizem aplikacije deluje podobno kot datotečni sistem, je najbolj smiseln drevesni pogled.

Ostane nam še aplikacija za urejanje in kreiranje prevodov (slika 4.9). Vsebuje dve glavni nastavitveni vozlišči: »actions« in »editor«. Vozlišče »actions« se drži istih pravil kot vozlišče »actions« pri brskalniških operacijah.

V vozlišču »**editor**« pa nastavimo dejanski obrazec za urejanje in vnos prevodov. V vozlišču »**actions**« referenciramo akcije po istem pravilu kot v orodni vrstici brskalniške operacije. V vozlišču »**nodeType**« pa definiramo tip vozlišča, ki ga aplikacija ureja. V vozlišču »**form**« nastavimo dejanski obrazec. Obrazci imajo lahko več zavihkov, v vsakem je lahko eden ali več

polj. Obrazec za prevode vsebuje tekstovna besedilna polja za ključ, angleško in slovensko vsebino. Če hočemo dodati dodatne jezike, tukaj ustvarimo dodatno polje. Vrsto polja določimo z lastnostjo »**class**« pri definiciji polja. Lastnost »**name**« pa določi ime lastnosti, v katero se bo zapisala vrednost polja. Ko obrazec shranimo, Magnolia v JCR samodejno zapiše novo vozlišče z ustreznimi vrednostmi.

4.5 Povzetek

V tem poglavju smo ustvarili modul za lokalizacijo statičnih vsebin. Magnoliino implementacijo i18n koncepta smo še dodatno razširili. Magnolia lahko namreč lokalizira samo celotno komponento ali celotno spletno stran. Naša implementacija pa lahko lokalizira tudi točno določeno besedilo v komponenti ali spletni strani. Rešitev omogoča tudi zelo enostavno izvedbo koncepta l10. Če želimo dodati nov jezik, potem v definiciji obrazca za vnos prevodov samo dodamo polje za jezik (slika 4.9).

V praksi si uporabniki za vsak jezik zgradijo svoje poddrevo spletnih strani, v katerega vnašajo ustrezno prevedene vsebine. Ta modul se uporablja za podporo lokalizaciji tam, kjer je res smiselno (primer z začetka poglavja).

Poglavje 5

Modul za podporo večim domenam

Vsebina aplikacije je lahko porazdeljena na veliko med seboj neodvisnih sklopov. Sklopi se lahko delijo po jezikih, po napravah, za katere so namenjeni (mobiteli, tablice, PC-ji), ali po logičnih sklopih (za otroke, za odrasle itd.). Če imajo uporabniki tako segmentirane vsebine, običajno registrirajo lastno domeno za vsak sklop. Ker prosta različica sistema Magnolia CMS ne podpira zaznavanja domen, je bilo treba razviti ustrezno rešitev.

Kot primer vzemimo prodajalca avtomobilov. Prodajalec ima registrirano domeno www.prodajamo-avtomobile.si. JCR-vozlišče vstopne strani se nahaja na poti
»/slovenscina/vstopna-stran«. Prodajalec se odloči, da bo organiziral nagradno igro, in registrira domeno www.zadeni-zastonj-avto.si. Pot do JCR-vozlišča z vsebino strani je »/slovenscina/nagradna-igra«. Ker se Magnolia ne zaveda domen, moramo vse obiskovalce usmeriti na »/slovenscina/vstopna-stran« ali na »/slovenscina/nagradna-igra«, ne glede na to, iz katere domene so dostopali do vsebine.

Preden bomo predstavili rešitev, bomo opisali, zakaj obstoječi načini reševanja niso primerni za naše potrebe. Sledil bo kratek opis HTTP-standarda, ker je razumevanje tega nujno za razumevanje delovanja modula.

5.1 Klasično reševanje več-domenskih vsebin

Običajno se take težave rešuje z uporabo navideznih domen. Podpirajo jih vsi posredovalni strežniki (angl. *proxy server*). Z uporabo navideznih domen lahko različne domene preusmerimo na en strežnik oziroma kar vsako domeno na različen URL na strežniku[8], kar povsem ustreza našim zahtevam.

Težava pri tem pristopu je, da ločimo nastavitve od aplikacije. Če namreč zamenjamo strežnik, na katerem teče aplikacija, moramo poiskati tudi vse nastavitve na strežnikih in jih ustrezno popraviti. Če projekt ni dobro dokumentiran, se kakšno domeno tudi hitro spregleda. Ker ima Magnolia vedno vsaj dva strežnika, nam to še dodatno oteži delo.

V tem poglavju smo zato razvili ustreznejšo rešitev, ki nastavitve ohrani na enem mestu znotraj aplikacije za vse instance. Obnaša se identično kot navidezne domene v posredovalnih strežnikih. Še ena prednost take rešitve je, da lahko navidezne domene vnaša uporabnik Magnoliinega administrativnega vmesnika, če ima za to pravice (običajno je to skrbnik aplikacije), ne pa administrator strežnikov.

5.2 Standard HTTP

HTTP je brezstanjski protokol za komunikacijo med odjemalci in strežniki[8]. Odjemalci na strežnik pošiljajo zahteve, strežniki pa odgovore. Na vsako zahtevo pride natanko en odgovor, če ni prišlo do zunanje napake pri povezavi.

5.2.1 HTTP zahteve

Zahteva je oblike

```
[metoda] [URL] [različica]
[glavini zapisi]

[jedro]
```

Dovoljene metode so podane v tabeli 5.1.

Metoda	Opis
GET	Pridobi vsebino
PUT	Dodaj vsebino
DELETE	Odstrani vsebino
POST	Posodobi vsebino
HEAD	Pridobi metapodatke o vsebini

Tabela 5.1: HTTP-metode[9]

Večina spletnih strani ne uporablja vseh metod, ampak samo GET za pridobivanje vsebin in POST za pošiljanje vsebin. URL je naslov vsebine na strežniku, različica pa pove, po kateri različici protokola je bila zahteva zgrajena, in posledično, katero različico pričakuje pri odgovoru. Glavini zapisi vsebujejo dodatne informacije. Najbolj pogoste so podane v tabeli 5.2.

V jedru zahteve so shranjene morebitne vsebine, ki jih je odjemalec poslal na strežnik. Vse zahteve so ASCII-besedilo.

5.2.2 HTTP odgovor

HTTP odgovori so oblike

```
[različica] [status] [razlog]
[glave]

[jedro]
```

Različica je identična kot pri HTTP-zahtevi. Status vsebuje informacijo o rezultatu zahteve, razlog pa je dodaten opis statusa. Statusi so trimestna števila in se delijo na kategorije, podane v tabeli 5.3.

Glavini zapisi odgovora vključujejo metapodatke o vsebini in služijo kot podlaga odjemalcu za prikazovanje in hranjenje vsebin.

Jedro je dejanska vsebina odgovora. Običajno je to HTML-koda spletne strani, lahko pa je kar koli. Tudi odgovor je ASCII-besedilo.

Glava	Opis
Host	Naslov domene, do katere smo dostopali. Edino obvezno polje.
Referer	URL na katerem je nastala zahteva.
User-Agent	Informacije o programu, ki je ustvaril zahtevo.
Accept	Pove katere vrste vsebine klient pričakuje (*MIME type*).
Accept-Language	Pove, kateremu jeziku odjemalec daje prednost.
Cookie	Informacije o piškotkih.
If-Modified-Since	Vsebuje datum, ko je odjemalec zadnjič dostopal do vsebine.
X-Forwarded-For	Če zahteva potuje preko posredovalnega strežnika, se sem zapiše izvirni vir zahteve.
Client-IP	Isto kot »X-Forwarded-For«. Obe imata isti pomen, katero bo uporabil posredovalni strežnik, je odvisno od njegove implementacije.

Tabela 5.2: Glavini zapisi HTTP-zahteve

Interval	Vrsta statusa	Opis
100–199	Opisni	Uporabljajo se za pogajanje med strežnikom in klientom.
200–299	Uspeh	Zahteva je bila uspešno obdelana
300–399	Preusmeritev	Rezultat zahteve je preusmeritev na neko drugo lokacijo
400–499	Napaka odjemalca	Napako je povzročila zahteva sama (napačen URL, napačni parametri, nima ustreznega dovoljenja za dostop).
500–599	Napaka strežnika	Do napake je prišlo pri procesiranju zahteve.

Tabela 5.3: Kategorije statusov v HTTP-odgovorih[9].

5.3 Izdelava rešitve

Za lažje razumevanje problema bomo najprej podali primer. V aplikacijo vstopi zahteva z domeno www.stran.si in URI-jem »/naslovnica/podstran«. Imamo večjezično aplikacijo in dejanska vsebina se nahaja v JCR-poti »/slovensko/naslovnica/podstran«. URL-ju moramo na začetek dodati »/slovensko«. Ko je HTML-vsebina že ustvarjena, pa moramo iz vseh povezav vsebini odstraniti »/slovensko«.

Želeno obnašanje rešitve:

- aplikacija mora glede na domeno HTTP-zahteve nastaviti ustrezno JCR-pot do vozlišča, ki se bo za to zahtevo obnašalo kot korensko vozlišče,
- ker Magnolia samodejno generira povezave do vsebin iz korenskega vozlišča JCR delovnega prostora »website«, moramo iz vseh povezav v HTTP-odgovoru odstraniti odvečen del poti.

V Magnolii gre vsaka HTTP-zahteva čez verigo filtrov[2], rezultat česar je HTTP-odgovor. Za ustrezno delovanje zato potrebujemo dva filtra. Filter, ki

bo URL-zahteve prepisal v želeno pot, in filter, ki bo vse povezave v odgovoru prepisal v želen URL-naslov na domeni. Prvi filter smo poimenovali vstopni filter, drugi pa izstopni.

Pri tem postopku se lahko pojavijo težave z vsebinami, ki niso odvisne od domene, ali pa se ne nahajajo v delovnem prostoru »website«. Primer takih vsebin so multimedijske datoteke. To težavo smo rešili tako, da smo v konfiguracijo dodali vnosa poti, ki jih filter ne bo upošteval.

Definirati je bilo treba tudi pravila za prepisovanje. Odločili smo se za prepisovanje s pomočjo regularnih izrazov. Pri tem je treba upoštevati še, da se nobena informacija iz URL-ja ne sme izgubiti. Če vzamemo primer iz začetka poglavja, potem potrebujemo regularni izraz, ki bo razbil URL-je »/(.*)«. Pravilo, ki je sestavilo URL v JCR-pot, je »/slovensko/$1«.

Splošna definicija prepisovanja:

- imamo regularni izraz, ki razkosa URL-je,
- vsak kos URL-ja dobi številčno oznako,
- oznaka najbolj levega kosa je 1, oznake naraščajo proti desni. Pri sestavljanju JCR-poti se nanje sklicujemo s »$[številka oznake]«.
- JCR-pot sestavimo iz kombinacije razbitih delov URL-ja in konstant.

5.3.1 Vstopni filter

Magnolia URL-je zahtev privzeto preslika najprej v delovni prostor, nato pa še v pot do JCR-vozlišča z ustrezno vsebino. Potrebujemo filter, ki bo URL-zahteve spremenil v JCR-pot glede na ustrezno domeno.

Tukaj je treba izpostaviti še, da lahko imamo globalne URL-je, ki so skupni za vse domene. Običajno so to URL-ji za dostop do statičnih vsebin, kot so slike, Javascript datoteke in CSS-datoteke.

Filter nastavimo v JCR-ju. Korensko vozlišče je »multiSiteIn«. Vsebuje dve lastnosti:

- »class«: polno ime javanskega razreda, kjer je implementirana logika,

- »enabled«: pove, ali ga Magnolia upošteva ali ne.

Za lažje razumevanje tega odstavka naj si bralec pomagas sliko 5.1. Korensko vozlišče ima tudi »bypasses« in »sites« naslednike. V vozlišču »**bypasses**« so shranjeni vsi URL-ji, ki jih filter ignorira ne glede na domeno. Vozlišče, ki definira ignorirani URL, vsebuje lastnosti »class« in »pattern«. »**Pattern**« vsebuje regularni izraz, ki določa URL, »class« pa vsebuje polno ime javanskega razreda, ki vsebuje logiko za pregledovanje URL-jev z regularnimi izrazi. Vozlišče »sites« pa vsebuje samo naslednike. Vsak naslednik predstavlja logično zaključeno vsebinsko celoto. V praksi to pomeni, da če imamo več domen, ki vsebujejo ista pravila za prepisovanje, jih lahko obravnavamo skupaj, namesto da bi podvajali nastavitve. V nadaljevanju opisa se bomo na take celote sklicevali z besedo »stran«, ker običajno vsebujejo le eno domeno.

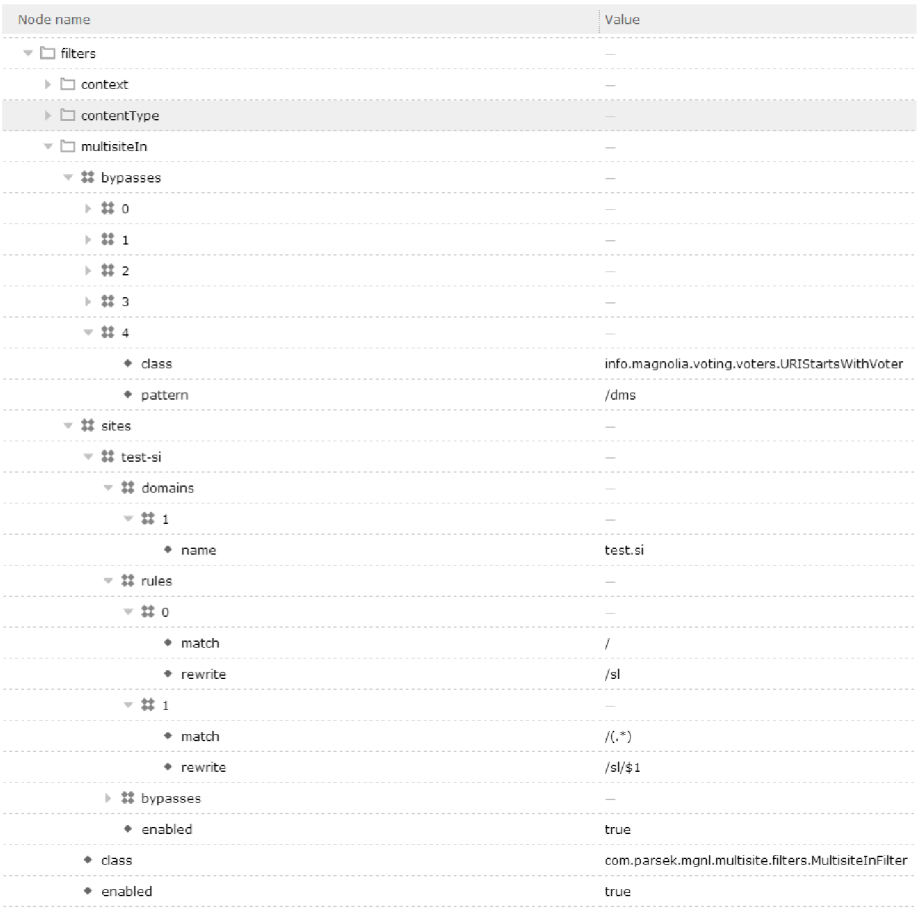
Vozlišče strani vsebuje lastnost »enabled« in tri naslednike. V naslednika »domains« so shranjene vse domene, za katere so veljavne nastavitve te strani. Vrednost v lastnosti »name« se mora natanko ujemati z vrednostjo v »host« glavi HTTP-zahteve, ki prihaja iz te domene. V vozlišču »rules« so definirana pravila za prepisovanje. Lastnost prepisovalnega pravila »match« je regularni izraz za razkosavanje URL-ja, lastnost »rewrite« pa je sestavljena pot. V naslednika »bypasses« so zapisani URL-ji, ki jih samo ta stran ne prepisuje. Nastavitve so identične kot za globalno vozlišče »bypasses«.

5.3.2 Izstopni filter

Razvoj izstopnega filtra se je izkazal za zahtevnejšega, kot je bil razvoj vstopnega filtra. Soočiti se je bilo treba z nekaj dodatnimi izzivi:

1. kam v verigo filtrov postaviti izstopni filter,
2. kaj vse je treba dejansko prepisati.

Reševanje prve točke se sliši kot trivialno opravilo. Ker prepisujemo končno vsebino, filter postavimo na konec verige, kjer samo popravimo vsebino že



Slika 5.1: Primer vstopnega filtra, ki obvladuje zahteve z domene test.si

ustvarjenega HTTP-odgovora. Izkaže se, da to ni tako preprosto. Privzeta implementacija ne omogoča niti branja vsebine HTTP-odgovora, ampak samo zapisovanje. Za rešitev te zagate smo bili primoran spisati nov razred za HTTP-odgovor, ki bo to omogočal. Tudi ko že imamo razred, filtra vseeno ne moremo postaviti na konec verige. Razlog tiči v tem, da imamo na koncu filtra samo HTTP-odgovor, brez konteksta, kaj se je vanj zapisalo. Če HTTP-odgovor vsebuje neko binarno datoteko, lahko ta datoteka zgolj naključno vsebuje zaporedje znakov, ki bi jih filter prepoznal, prepisal in s tem uničil datoteko. Vrsto datoteke bi lahko prebrali iz glave HTTP-odgovora, vendar je veliko lažje, če za to uporabimo pristop, kot ga bomo opisali spodaj.

Izstopni filter postavimo tik pred prvi filter, ki začne pisati HTTP-odgovor. V izstopnem filtru zamenjamo privzeti HTTP-odgovor z našim. Ko se vsi filtri za izstopnim filtrom izvedejo, se aplikacija spet vrne v izstopni filter. Tam lahko sedaj s pomočjo Magnoliinega API-ja ugotovimo, za kakšno vsebino je šlo in, če je potrebno, jo prepišemo. Na koncu preberemo vso vsebino iz naše implementacije HTTP-odgovora in jo zapišemo v privzeti HTTP-odgovor, ki tudi zapusti naš filter. Na prvi pogled je tudi vprašanje iz druge točke trivialno. Prepišemo vse, kar je v jedru HTTP-odgovora in je v skladu z regularnim izrazom, ki je podan v filtru. Ni čisto tako. Prvo težavo predstavljajo datoteke, kar smo opisali v prejšnjem odstavku. Posebej pozorni pa moramo biti tudi na odgovore, ki imajo kodo statusa med 300 in 399. To so odgovori, ki odjemalcu sporočajo, da je iskana vsebina na drugi lokaciji. Ta vsebina je podana z URL-jem v glavi HTTP-odgovora, ki jo moramo upoštevati tudi pri prepisovanju.

Korensko vozlišče izstopnega filtra se imenuje »multisiteOut« in se nastavlja identično kot za vstopni filter. Primer enostavne nastavitve je podan v sliki 5.2.

Node name	Value
multisiteOut	–
bypasses	–
0	–
1	–
2	–
3	–
4	–
sites	–
test-si	–
domains	–
1	–
name	test.si
rules	–
0	–
match	(^|href=")/sl/(.*?)("|$)
rewrite	$1/$2$3
bypasses	–
0	–
pattern	/.magnolia
1	–
2	–
3	–
enabled	true
class	si.spica.web.filters.FormMultisiteOutFilter
enabled	true

Slika 5.2: Primer izstopnega filtra, ki obvladuje zahteve z domene test.si

5.4 Povzetek

V tem poglavju smo opisali razvoj modula, ki oponaša funkcionalnost navideznih domen v posredovalnih strežnikih. Na videz smo odkrili toplo vodo, vendar to ni čisto res. Vse nastavitve za navidezne domene smo združili na eno mesto znotraj same aplikacije. To nam omogoča prosto prenosljivost aplikacije med posameznimi strežniki, brez da bi jih bilo treba še dodatno nastavljati. Omogoči nam tudi boljšo odzivnost na zahteve uporabnikov aplikacije in boljšo varnost strežniške infrastrukture. Vnašalec vsebine lahko sedaj registrira domeno in vnese vsebino. Skrbnik aplikacije nato v Magnoliinem administrativnem vmesniku enostavno vnese podatke o domeni, za kar ne potrebuje neposrednega dostopa do strežnikov.

V prihodnosti bi lahko razvili še uporabniški vmesnik za enostavno vnašanje navideznih domen (podobno kot v poglavju 4) in s tem bi lahko vnašalci vsebine opravili vse potrebno delo.

Poglavje 6

Shranjevanje podatkov iz spletnih obrazcev

Spletne strani že dolgo ne samo prikazujejo vsebin, ampak jih ljudje uporabljamo tudi za vnos podatkov. Lahko rešujemo samo smešne ankete, lahko pa tudi izvajamo bančne transakcije. V tem poglavju bomo pogledali in tudi rešili težavo, s katero se sreča skoraj vsak uporabnik sistema Magnolia CMS.

Kot smo že napisali, Magnolia teče v dveh instancah. Avtorsko instanco uporabljajo upravljavci vsebine, medtem ko obiskovalci spletnega mesta dostopajo do javne instance. Vsebina se torej prenaša iz avtorske instance v javno, nikoli obratno. To pa nam predstavlja težave ravno pri sprejemanju uporabnikovih podatkov. Uporabnik namreč pošlje podatke v javno instanco, kjer "obtičijo", saj do njih ni mogoče dostopati. Avtorji Magnolie se tega zavedajo, zato podatkov niti ne shranjujejo. Magnolija ima privzet vmesnik, po katerem pošilja podatke neposredno na nek vnaprej nastavljen e-naslov. Vmesnik deluje, ampak že konceptualno trpi za nekaj dokaj resnimi pomanjkljivostmi:

1. če se spremeni prejemnikov e-naslov, bodo od takrat naprej vsi podatki izgubljeni,
2. lahko pride nov administrator z drugačnim e-naslovom, ki ne bo dobival podatkov,

3. če spletni strežnik začasno ali trajno ne deluje, se bodo vsi podatki v tem času izgubili,

4. prejemnik podatkov lahko po nesreči izbriše sporočilo in s tem trajno izgubi podatke,

5. težko je iz e-pošte zbirati podatke, še posebej, če gre za večjo količino,

6. če veliko uporabnikov izpolnjuje obrazce v krajšem časovnem obdobju, se lahko preobremeni poštni strežnik,

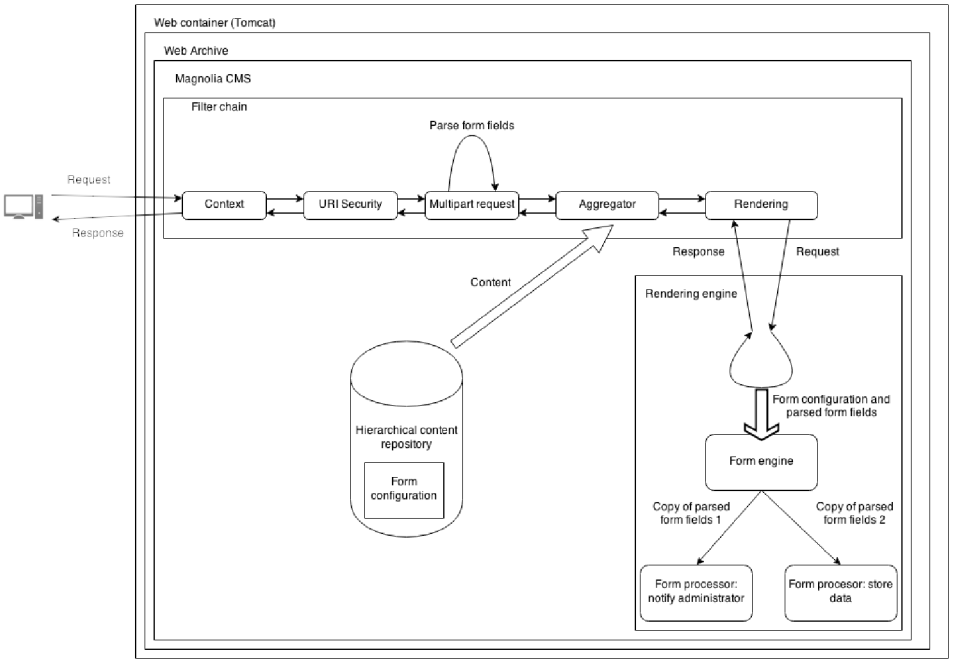
7. sporočila lahko začnejo prestrezati filtri za neželeno e-pošto (angl. *spam*).

Za rešitev vseh naštetih težav je treba implementirati mehanizem, ki bo samodejno in trajno shranil podatke.

Vsaj v osnovi je treba razumeti tudi mehanizem obrazcev v Magnolii. Obrazci so v Magnolii navadna vsebinska komponenta. Shranjujejo se v »website« delovni prostor kot naslednik vozlišča strani, na kateri se nahajajo. Komponenti obrazec lahko potem poljubno dodajamo in brišemo vsa ustrezna polja za pošiljanje podatkov. Magnolia zato ubere malo drugačen sistem za določanje, iz katerega obrazca so podatki v HTTP-zahtevi (slika 6.1).

HTML obrazci že po standardu vsebujejo parameter action, ki določa URL na strežniku, kjer se opravi procesiranje podatkov. Magnolija je tukaj ubrala drugačen pristop. Vsak obrazec dobi unikaten identifikator. Vrednost identifikatorja se nato izriše v HTML obrazcu, kot skrito - za uporabnika nevidno polje. Magnolija v contentType filtru pregleda, če zahteva vsebuje obrazec. Če ga vsebuje in če ima veljaven identifikator, jo procesira posebej, drugače jo normalno spusti čez ostale filtre.

Magnolija samodejno delegira obdelavo podatkov v tako imenovane procesorje. Vsak procesor ima poln dostop do vseh podatkov v obrazcu in z njimi lahko počne karkoli, brez da bi vplival na ostale procesorje[2]. Dostop ima tudi no nastavitvenega vozlišča v JCR bazi. Že ob namestitvi modula za obrazce imamo na volji tri procesorje, ki nam lahko služijo kot referenca



Slika 6.1: Shema obdelave HTTP-zahteve, ki vsebuje podatke obrazca. Obrazec vsebuje dva procesorja.

za ostale:HTML-obrazci že po standardu vsebujejo parameter »action«, ki določa URL na strežniku, kjer se opravi procesiranje podatkov. Magnolija je tukaj ubrala drugačen pristop. Vsak obrazec dobi unikaten identifikator. Vrednost identifikatorja se nato izriše v HTML-obrazcu, kot skrito – za uporabnika nevidno polje. Magnolia v »contentType« filtru pregleda, ali zahteva vsebuje obrazec. Če ga vsebuje in če ima veljaven identifikator, jo procesira posebej, drugače jo normalno spusti čez ostale filtre.

Magnolia obdelavo podatkov že na začetku delegira v tako imenovane procesorje. Vsak procesor ima poln dostop do vseh podatkov v obrazcih in z njimi lahko počne kar koli, brez da bi vplival na ostale procesorje. Dostop ima tudi do nastavitvenega vozlišča v JCR-bazi. Že ob namestitvi modula za obrazce imamo na volji tri procesorje, ki nam lahko služijo kot referenca za ostale:

- »TrackEmailProcessor«: uporabimo ga, če želimo nekemu administratorju sporočiti, da je bil obrazec izpolnjen,
- »SendContactEMailProcessor«: ta procesor zapakira vsebino obrazca v e-poštno sporočilo. Če je obrazec vseboval tudi datoteke, jih doda kot priponke. Predlogo e-sporočila lahko poljubno nastavimo ob kreiranju obrazca »Sporočilo«, ki ga potem pošlje prejemnikom, ki so bili nastavljeni ob kreiranju obrazca,
- »SendConfirmationEMailProcessor«: deluje na isti način kot »SendContactEMailProcessor«, le da se običajno pošlje uporabniku, ki je izpolnil obrazec.

6.1 Izbira rešitve

Najzahtevnejši del načrtovanja se je odločiti, kam naj se podatki shranijo. Imamo več možnosti:

- shranijo se v podatkovno bazo javne instance,

- shranijo se v podatkovno bazo avtorske instance,
- shranijo se v posebno podatkovno bazo.

Vsak pristop ima svoje dobre in slabe strani. Pri prvih točkah kršimo ločenost avtorske in javne instance, vendar ne večamo števila instanc podatkovnih baz. Če izberemo katero iz prvih dveh rešitev, se moramo zavedati tudi, da smo primorati izbrati JCR podatkovno bazo. Z bazo samo ni nič narobe, vendar lahko nastanejo praktične težave, če tako rešitev predamo v vzdrževanje zunanjemu izvajalcu. Če JCR primerjamo z razširjenostjo ostalih baz, predvsem relacijskih, opazimo, da je precej nišna tehnologija. Za shranjevanje podatkov smo zato uporabili rešitev iz tretje točke. Ločena baza nam namreč omogoča, da lahko podatke beremo od kjer koli. Za varnostne nastavitve lahko poskrbimo na ravni baze. Tako rešitev lahko vzamemo tudi kot primer zelo enostavne integracije Magnolie z nekim zunanjim sistemom.

6.2 Načrtovanje podatkovne baze

Celotna rešitev je bila spisana v JPA *(Java Persistence API)*. Javin API za komuniciranje z relacijskimi podatkovnimi bazami. Če ga pravilno nastavimo, je zmožen sam upravljati z zaledno bazo, tudi ustvarjanje relacij iz razredov[10]. Mi smo v zaledju uporabljali »PostgreSQL«, zato v nadaljevanju pišemo tudi z vidika »PostgreSQL« terminologije.

Po specifikaciji HTML-obrazcev lahko podatke razdelimo v tri logične kategorije:

- besedilni,
- logični *(true* ali *false)*,
- datoteke.

Na prvi pogled imamo tri domene, »boolean« (logična), »varchar« (besedilo) in »bytea« (datoteke). Po premisleku pa lahko izločimo logično domeno, ker

jo lahko v procesorju zapišemo kot besedilo *(true, false)*. Ostaneta nam samo še dve domeni.

Bralec se morebiti sprašuje, zakaj nismo izbrali NoSQL podatkovne baze, ker bi se tak načrt lepo preslikal vanjo. Tukaj se nam predvsem ponuja dokumentna podatkovna baza. Dokument bi bil poljuben zapis prejetih podatkov. Paziti bi morali le, da bi vsi dokumenti istega obrazca imeli nek skupen atribut. Dobra izbira za tak atribut bi bila pot do JCR-vozlišča, kjer je definiran obrazec. Potem bi lahko naredili poizvedbo po tem atributu in dobili vse dokumente s podatki iz enega obrazca. Zakaj smo se temu navkljub odločili za relacijsko bazo, sem nakazal že v poglavju 6.1 in je povsem praktične narave. Mi Magnolio uporabljamo predvsem za zahtevnejše projekte, ki zahtevajo integracijo z že obstoječimi sistemi. Ti za enkrat vsebujejo zgolj relacijske podatkovne baze.

Naslednja odločitev je, ali lahko domene zapisujemo v isto relacijo. Tehnološke omejitve ni. Naredimo relacijo podatek, ki vsebuje stolpca »besedilo« in »datoteka«. Takoj opazimo, da se ta dva stolpca medsebojno izključujeta. Besedilni vnos ne bo nikoli imel datoteke in obratno. Če upoštevamo, da bomo želeli shraniti tudi kakšne metapodatke, se kompleksnost take relacije hitro povečuje. Ustrezna rešitev je rešitev z dvema relacijama. Posebej bomo imeli relacijo za besedilne lastnosti in posebej za datoteke.

Relacija za besedilne atribute

Poleg vrednosti atributa potrebujemo vsaj še ime atributa. Odločili smo se za dodaten atribut relacije, ki nam pove, na katerem mestu je bil atribut v HTTP-zahtevi ob pošiljanju obrazca. Imamo torej stolpca »atribute«:

- »name«: ime atributa, domena »varchar«,
- »value«: vrednost atributa, domena »varchar«,
- »order_idx«: mesto atributa, domena »bigint«.

Relacija vsebuje še dva dodatna stolpca, s katerima JPA samodejno upravlja:

- »id«: primarni ključ,
- »form_data_id«: tuj ključ do relacije za obrazec.

Relacija za datoteke

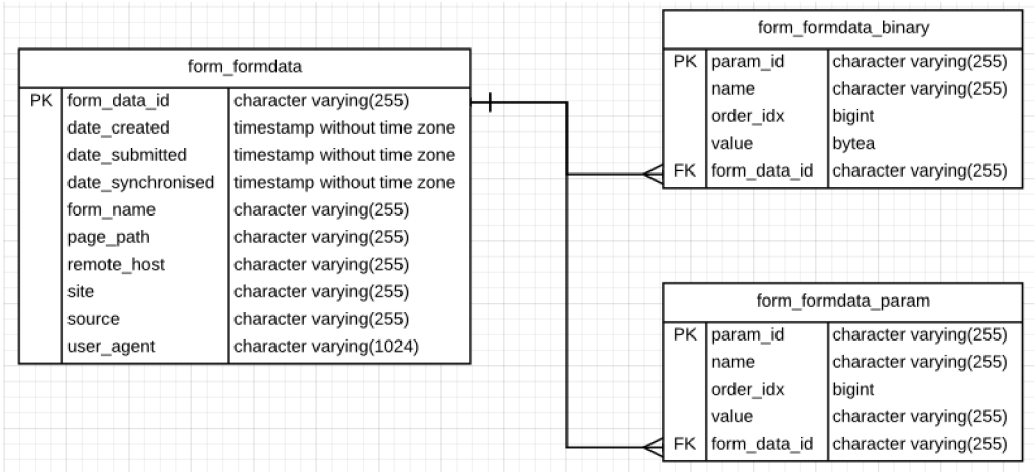
Relacija je praktično identična relaciji za besedilne atribute, edina razlika je v domeni stolpca »value«. Domena tega je »bytea« – polje bajtov.

Relacija za splošne podatke obrazca

V to relacijo se shranjujejo podatki o obrazcu. Lahko si jo predstavljamo tudi kot relacijo z metapodatki, ki jih vsebuje vsaka HTTP-zahteva s podatki iz obrazca. Vsak atribut (stolpec) nam predstavlja eno vrsto metapodatka. Stolpec »**date_created**« nam pove, kdaj je bil generiran zapis v relacijo. V stolpec »**date_submitted**« se shrani čas, ob katerem je HTTP-zahteva vstopila v sistem. V kombinaciji s stolpcem »date_created« nam služi za odkrivanje ozkih grl pri procesiranju podatkov obrazca. V stolpec »**form_name**« se zapiše ime obrazca, ki ga administrator vnese ob kreiranju komponente. V »**page_path**« se zapiše JCR-pot do vozlišča, ki predstavlja spletno stran, v kateri se nahaja obrazec. Stolpec »remote_host« vsebuje lokacijo, ki je ustvarila zahtevo. Podatek se bere iz glave HTTP-zahteve. Java EE že pozna metode za pridobivanje teh podatkov, vendar ne deluje, če so med uporabnikom in strežnikom postavljeni dodatni posredovalni strežniki. V tem primeru zapiše njihove lokacije namesto uporabnikovih. Zato najprej pogledamo, ali v HTTP-zahtevi obstajata glavi »Client-ip« in »X-Forwared-For«. Ti dve glavi načeloma vedno vsebujeta pravo uporabnikovo lokacijo. Če imamo večdomensko aplikacijo, se v atribut »site« zapiše ime korenskega vozlišča domene, kjer se nahaja stran z obrazcem. V »**source**« se zapiše ime strežnika, ki je obdelal HTPP-zahtevo. V »**user_agent**« shranimo informacijo o spletnem odjemalcu, ki je zahtevo ustvaril.

JPA nam samodejno ustvari stolpec »id«. Predstavlja nam primarni ključ relacije. JPA samodejno upravlja z njim.

Rezultat načrtovanja je torej shema s tremi relacijami, podana v sliki



Slika 6.2: Podatkovni model

6.2. Kot smo že omenili, JPA samodejno upravlja s primarnimi in tujimi ključi. Relacijama »form_formdata_binary« in »form_formdata_param« samodejno ustvari tuji ključ »form_data_id« in ga poveže z atributom id relacije »form_formdata«.

6.3 Implementacija

V Javi spišemo ustrezne razrede, ki jih JPA nato samodejno poveže z relacijami [10]. Za vso logiko poskrbi procesorski razred. Magnoliin API mu samodejno dodeli dostop do vseh potrebnih podatkov. V razredu samo ustvarimo ustrezne atribute, ki jih JPA samodejno shrani v pravilne relacije. Komponenti »obrazec« v JCR-ju nato dodamo še ustrezen procesor. Tukaj opisana rešitev lahko shranjuje vse vrste obrazcev.

Do shranjenih podatkov lahko dostopamo preko poljubnega okolja za prikaz podatkov v relacijskih bazah.

6.4 Povzetek

Modul je namenjen predvsem integraciji z že obstoječimi sistemi, zato nima možnosti prikaza zajetih podatkov. Če si uporabnik aplikacije želi ažurno obveščanje o prejetih podatkih, si lahko vedno nastavi sporočanje preko e-pošte. Ker se potreba po tem običajno pojavi pri kratkotrajnih vsebinah (nagradne igre, promocije itd.), pomisleki glede obveščanja preko e-pošte niso tako pereč problem. Po potrebi bi lahko v Magnoliinem administrativnem vmesniku razvili aplikacijo, ki bi vse podatke za posamezen obrazec izvozila v tabelarično datoteko (csv, xls itd.).

Dodaten pomislek je tudi varnost te rešitve. Omenili smo že, da je za varnost poskrbljeno na ravni podatkovne baze. Opisana implementacija modula tudi nima možnosti branja podatkov, zato ni bojazni, da bi podatki lahko izginili. Če bi želeli tudi brati podatke, bi to storili preko Magnoliinega administrativnega vmesnika, ki potrebuje prijavo z uporabniškim imenom in geslom.

Poglavje 7

Zaključek

V diplomski nalogi smo na hitro predstavili sistem Magnolia CMS, drevesne podatkovne baze in funkcionalne razširitve za Magnolio.

Po izkušnjah pri delu z Magnolio je ta primerna predvsem za srednje velike projekte. Njena neposredna konkurenta sta Drupal in Joomla. Podobno kot ta dva sistema Magnolia razvijalca v ničemer ne omejuje pri izdelavi HTML-strani. HTML-razrez lahko sami implementiramo, kot si želimo, kar nam ne omogočata niti WordPress niti LifeRay. Vgrajen je tudi odličen sistem za omejevanje dostopa uporabnikom, za kar moramo pri Drupalu in Joomli poskrbeti sami. Njena učna krivulja tudi ni tako strma kot pri LifeRayu. V primerjavi z njim porabi tudi precej manj resursov. Najbolj se izkaže pri vnašanju vsebin na spletne strani, saj vnašalec takoj vidi vsako spremembo točno tako, kot bo videti na strani.

Ima pa tudi določene težave. Zaradi ločenosti med avtorsko in javno instanco ima težave s shranjevanjem podatkov, ki jih pošiljajo obiskovalci. V tem diplomskem delu smo razvili tudi modul, ki razrešuje točno to težavo (poglavje 6). Privzeta implementacija za lokalizacijo vsebin se tudi izkaže za rahlo pomanjkljivo. Enostavno lahko lokaliziramo samo vsebine, ki jih uporabnik vnaša neposredno na spletno stran. V poglavju 4 smo zato razvili modul, ki omogoča lokaliziranje vsebin, nad katerimi uporabnik drugače ne bi imel nadzora oziroma ne bi bilo smiselno, da jih vnese več kot enkrat za

vsak jezik. V poglavju 5 pa smo razvili modul, ki Magnolii omogoči, da se začne zavedati domen. To nam omogoči nastavljanje navideznih domen znotraj same aplikacije. Ker Magnolia vedno teče vsaj na dveh strežnikih, to močno olajša nastavljanje strežniške infrastrukture.

Upamo, da smo z njimi uspešno pokazali, kako modularna je zasnova sistema in kako enostavno je dodajati ali spreminjati že obstoječe module ali celo osnovne funkcionalnosti.

Glavno vprašanje smo prihranili za konec. Ali naj za svoje projekte uporabimo Magnolio? Odgovor je pritrdilen, vendar tega ne smemo narediti na slepo. Kot smo posredno že omenili, Magnolia ni primerna za manjše projekte, ker je razvoj v primerjavi z WordPressom časovno potraten. Veliki projekti s seboj vedno prinesejo mnogo specifičnih zahtev. Če tak projekt delamo z Magnolio, zelo hitro ugotovimo, da več časa razširjamo in prepisujemo že obstoječe funkcionalnosti, kot pa pišemo aplikacijo.

Glavna konkurenta Magnolie sta torej Drupal in Joomla, izbiro pa lahko skrčimo na izbiro med javanskim in PHP-jevim ekosistemom. To izbiro prepuščamo bralcu.

Literatura

[1] J. Anano, "Content managment system," 2009.

[2] (2014) Magnolia 5 documentation. [Online]. Available: http://documentation.magnolia-cms.com/display/DOCS/Magnolia+5+Documentation

[3] P. C. Kanellakis, "Handbook of theoretical computer science (vol. b)," J. van Leeuwen, Ed. Cambridge, MA, USA: MIT Press, 1990, ch. Elements of Relational Database Theory, pp. 1073–1156. [Online]. Available: http://dl.acm.org/citation.cfm?id=114891.114908

[4] B. Chapuis, "JCR or RDBMS," *Day Software AG*, 2008.

[5] K. Ruohonen, *Graph Theory*. Tampere University of Technology, 2008.

[6] (2009) JCR v2.0 specification. [Online]. Available: http://www.day.com/maven/jcr/2.0/

[7] A. Phillips and M. Davis. (2009) Tags for identifying languages. [Online]. Available: http://tools.ietf.org/html/bcp47

[8] S. Allen, *HTTP Succinctly*. Syncfusion, Inc., 2012.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. (1999) Hypertext transfer protocol – HTTP/1.1. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html

[10] (2009) JSR 317: JavaTM persistence 2.0. [Online]. Available: https://jcp.org/aboutJava/communityprocess/final/jsr317/